A Refinement based methodology for software process modeling

Fahad Rafique Golra

N° d'ordre : 2014telb0304

Sous le sceau de l'Université européenne de Bretagne

Télécom Bretagne

En habilitation conjointe avec l'Université de Rennes 1

Ecole Doctorale – MATISSE

A REFINEMENT BASED METHODOLOGY FOR SOFTWARE PROCESS MODELING

Thèse de Doctorat

Mention : Informatique

Présentée par **Fahad Rafique – GOLRA**

Département : Informatique

Laboratoire : IRISA

Directeur de thèse : Antoine Beugnard

Soutenue le 08 janvier 2014

Essentially, all models are wrong, but some are useful.

George E. P. Box

Acknowledgments

I would like to gratefully and sincerely thank Fabien Dagnat for his guidance, understanding, patience, and most importantly, his friendship during my graduate studies at Telecom Bretagne. His mentorship was paramount in providing a well-rounded experience consistent to my long-term career goals. He encouraged me to not only grow my expertise in my research domain but also as an independent thinker. Not only did he ignite my imaginations, he also stimulated the thirst and the desire to know. I would like to thank Antoine Beugnard for his constant guidance and the liberty to explore the domain with his constant support. I am not sure many graduate students are given the opportunity to develop their own individuality and self-sufficiency by being allowed to work with such independence.

I would also like to thank all of the members of PASS research team at Telecom Bretagne, especially Maria-Teresa Segarra and Julien Mallet and all other fellow doctoral/post-doctoral researchers for giving me the opportunity to discuss and evolve this research project. Additionally, I am very grateful for the friendship of all of the members of his research group, especially Aladin, Jean Baptiste, Thang, Ali and An Phung, with whom I worked closely and puzzled over many similar problems. I would also like to thank my colleagues in office and Serge Garlatti who provided for some much needed humor and entertainment in what could have otherwise been a somewhat stressful laboratory environment. I specially want to thank Armelle Lannuzel, who has helped me through all administrative procedures during my stay. I would like to thank the members of my defense jury for their input, valuable discussions and proposed corrections, that shall guide me further to improve my professional skills.

Finally, and most importantly, I would like to thank my wife Asfia Firduas. Her support, encouragement, quiet patience and unwavering love were undeniably the foundations upon which the past six years of my life have been built. I thank my parents, Rafique Golra and Nusrat Shaheen, for their faith in me and allowing me to be as ambitious as I wanted. It was under their watchful eye that I gained so much drive and an ability to tackle challenges head on. Also, I thank my brothers Asad and Shoaib who provided me with unending encouragement and support.

Abstract

There is an increasing trend to consider the processes of an organization as one of its highly valuable assets. Processes are the reusable assets of an organization which define the procedures of routine working for accomplishing its goals. The software industry has the potential to become one of the most internationally dispersed high-tech industry. With growing importance of software and services sector, standardization of processes is also becoming crucial to maintain credibility in the market. Software development processes follow a lifecycle that is very similar to the software development lifecycle. Similarly, multiple phases of a process development lifecycle follow an iterative/incremental approach that leads to continuous process improvement. This incremental approach calls for a refinement based strategy to develop, execute and maintain software development processes.

This thesis develops a conceptual foundation for refinement based development of software processes keeping in view the precise requirements for each individual phase of process development lifecycle. It exploits model driven engineering to present a multi-metamodel framework for the development of software processes, where each metamodel corresponds to a different phase of a process. A process undergoes a series of refinements till it is enriched with execution capabilities. Keeping in view the need to comply with the adopted standards, the architecture of process modeling approach exploits the concept of abstraction. This mechanism also caters for special circumstances where a software enterprise needs to follow multiple process standards for the same project.

On the basis of the insights gained from the examination of contemporary offerings in this domain, the proposed process modeling framework tends to foster an architecture that is developed around the concepts of “design by contract" and “design for reuse". This allows to develop a process model that is modular in structure and guarantees the correctness of interactions between the constituent activities. Separation of concerns being the motivation, data-flow within a process is handled at a different abstraction level than the control-flow. Conformance between these levels allows to offer a bi-layered architecture that handles the flow of data through an underlying event management system. An assessment of the capabilities of the proposed approach is provided through a comprehensive patterns-based analysis, which allows a direct comparison of its functionality with other process modeling approaches.

Keywords : Process Modeling, Software Development, Metamodels, Model Driven Architecture, Process Refinement, Component-oriented Process Modeling Framework (CPMF)

Résumé

Il y a une tendance croissante à considérer les processus d'une organisation comme l'une de ses grandes forces. Les processus sont des ressources réutilisables d'une organisation qui définissent les procédures de travail pour la réalisation de ses objectifs. Avec l'importance croissante du secteur des logiciels et des services, la standardisation des processus devient indispensable pour maintenir sa crédibilité. Le développement de processus suit un cycle de vie très similaire à celui du développement logiciel. Par exemple, il se compose de plusieurs phases et suit une approche incrémentale qui mène à son amélioration continue. Cette approche incrémentale peut être complétée par une stratégie basée sur le raffinement pour développer, exécuter et maintenir les processus de développement de logiciels.

Cette thèse propose une base conceptuelle pour le développement de processus logiciels par raffinement, sans perdre de vue les exigences spécifiques de chaque phase du cycle de vie d'un tel processus. Elle utilise l'ingénierie dirigée par les modèles pour présenter un ensemble de méta-modèles pour le développement de processus logiciels où chaque méta-modèle correspond à une phase différente d'un processus (spécification, implémentation et instanciation). Le modèle d'un processus traverse une série de raffinement jusqu'à ce qu'elle soit enrichie par des capacités d'exécution. Le développement d'un interpréteur permet d'exécuter ce modèle. Il donne la possibilité de relier les modèles des differentes phases par des liens de traçabilité. Les intervenants peuvent interagir avec le processus en exécution à l'aide d'une interface de supervision. Un niveau de variabilité incluse dans les modèles de processus permet leur adaptation pendant l'exécution. Tout en prenant en compte la nécessité de se conformer aux standards adoptés par l'organisation, l'architecture de l'approche de modélisation proposée exploite le concept d'abstraction en s'inspirant de la notion de composant logiciel pour aider à la réutilisation de modèles de processus. Notre méthode est également prévue pour les entreprises qui veulent suivre plusieurs standards pour le même projet.

Sur la base des connaissances acquises grâce à l'étude des langages de modélisation actuels du domaine, le cadre proposé pour la modélisation de processus présente une architecture qui se développe autour des concepts de «conception par contrat» et «conception pour et par la réutilisation». Ceci permet de construire un modèle de processus qui a une structure modulaire et garantit la correction des interactions entre des activités constituantes. Afin de favoriser la séparation des préoccupations, les flux de données au sein d'un processus sont gérés à un niveau d'abstraction différent de celui des flux de contrôle. La conformité entre ces deux niveaux permet d'offrir une architecture bicouche. Le flux de données lors de l'exécution est assuré par un système de gestion d'événements. Une évaluation des capacités de l'approche proposée est fournie par une analyse basée sur l'ensemble des «workflow patterns». Cela permet une comparaison directe de ses capacités avec d'autres approches de modélisation de processus.

Mots-clés : Processus de développement logiciel, Modélisation de processus, Activité, Ingénierie Dirigée par les Modèles, Raffinement

Contents

Résumé en français

Sommaire

Tout comme les systèmes logiciels, les processus logiciels sont basés sur la notion de cycle de vie, où chaque étape du développement des concepts différents à abordé. Chaque phase d'un processus logiciel organise différentes questions liées à son degré de maturité. Fuggetta définit un processus logiciel comme un ensemble cohérent de politiques, de structures organisationnelles, de technologies, de procédures et d'artefacts qui sont nécessaires pour concevoir, développer, déployer et maintenir un logiciel [Fuggetta 00]. Mais nous avons tendance à penser comme Osterweil, les processus partagent la nature et la complexité des systèmes logiciels et doivent être traités de la même manière [Osterweil 87]. Ainsi, nous pensons que les processus logiciels doivent être conçus (spécifications), développés (implémentation), déployés (instanciation) et maintenus (surveillance). Pendant que le cycle de vie d'un processus logiciel avance, l'accent sur les questions liées au processus changent également. Par exemple, en phase de spécification l'accent est mis sur la définition des artefacts qui seraient traités / développés au cours de l'activité. D'autre part, pour l'exécution d'un processus le focus se déplace sur les échéances temporelles qu'un artefact doit respecter. Pendant la phase d'exécution, la mise au point de l'activité ne correspond pas au choix de ses entrées et sorties, il est plutôt lié au «comment» et au «quand».

Les modèles sont utilisés pour différentes raisons : prédire les propriétés d'un système, raisonner sur son comportement, communiquer entre les différents espaces techniques, etc. Dans le domaine d'informatique, ils sont actuellement utilisés pour automatiser le développement du système d'informatique [Kent 02]. Différents aspects d'un système peuvent être considérés comme pertinents à différents moments, et chacun de ces aspects peu être modélisé en utilisant des éléments de modèles différents. Dans de tels cas, certaines règles peuvent raffiner ces modèles pour les transformer en une autre perspective. Ces règles peuvent également être utilisés pour enrichir ces

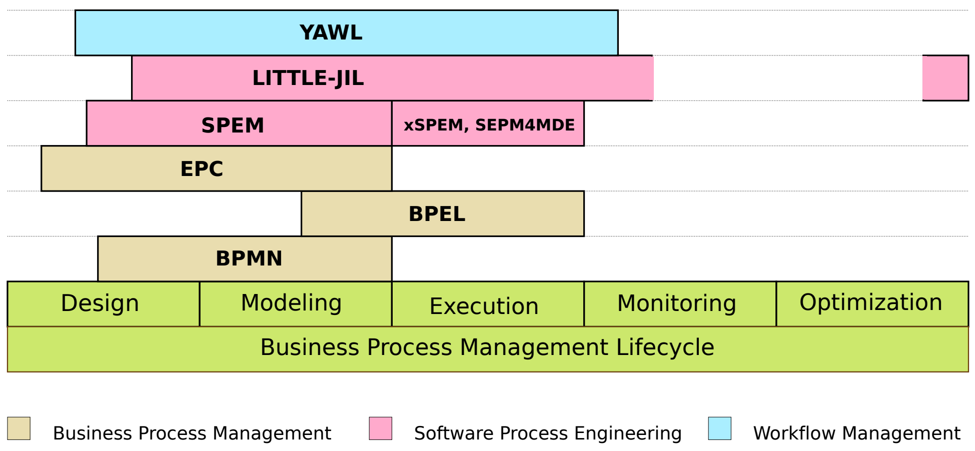


FIGURE 1 – La couverture du cycle de vie des processus

modèles avec des informations afin de les raffiner. Les modèles et les transformations de modèles forment le noyau de l'ingénierie dirigée par les modèles (IDM). Les récents progrès dans le domaine de l'IDM ont permis la construction d'outils, qui peuvent être utilisés par des architectes système et des développeurs pour modéliser les différentes perspectives d'une système et les relier directement au code dans un langage de programmation d'une plate-forme particulière [Jouault 06]. L'utilisation de modèles en génie des processus offre les mêmes avantages. Les processus peuvent être développés en utilisant des approches de modélisation et affinés au fil du temps, selon les cycles de vie de développement de processus. Les techniques d'ingénierie dirigée par les modèles peuvent être utilisés pour le raffinement des modèles de processus.

1 État de l'art

Les langages de modélisation de processus présentent deux problèmes. Tout d'abord, ils semblent ignorer l'importance d'une approche cohérente qui gère le processus à toutes les étapes de son cycle de vie. Soit un modèle de processus unique représente un processus dans toutes les phases (par exemple lors de la phase de spécification et la phase d'implémentation) ou alors il doit être transformé en une approche complètement différente pour l'exécution. Par exemple BPMN [OMG 11] représente des processus à toutes les étapes de développement en utilisant une seule notation, mais il n'offre pas la possibilité de les exécuter. Par conséquent, les développeurs de processus sont obligés de transformer ces modèles en BPEL [OASIS 07] pour les exécuter. La couverture du cycle de vie des processus de développement pour les approches différentes dans l'état de l'art est illustrée à la figure 1.

Deuxièmement, la plupart des approches se concentrent sur le flux des activités pour définir l'ordre de l'exécution (présumé). Certaines approches utilisent des mécanismes basée sur des événements pour induire une réactivité, mais leur attention reste sur le flux des activités [Adams 05]. Des approches telles que l'*Event Process Chains* (EPC) utilisent les deux types d'entrées pour les activités (événements et artefacts), qui encombrent le modèle de processus [van der Aalst 99]. Avec le flux de données et les flux de contrôle dans un processus en même temps, il est difficile de conceptualiser les interactions d'une activité à son contexte.

Un désavantage général de l'utilisation des langages de modélisation de processus basés sur les flux est le manque de dynamisme et de réactivité dans le modèle de processus. Cependant, en utilisant la notion d'événements et la gestion des exceptions, BPMN propose de développer des modèles de processus réactifs. D'autre part, SPEM ne prend pas en compte les événements limitant ainsi les concepteurs à une approche proactive. Certaines des extensions de SPEM, comme xSPEM, ajoutent cette capacité et les modèles de processus sont enrichis avec des contrôles réactifs [OMG 08]. D'autre part, les événements sont une notion centrale pour les interactions entre les activités dans les approches basées sur les événements. Si deux activités EPC doivent interagir, elles doivent utiliser des événements. Les *Workflow Management Systems* (WfMS) placent également la notion d'événements au ceur du modèle d'interaction, qui se traduit par la réactivité d'ensemble des modèles de processus [Hollingsworth 04].

La nature même d'un processus est hiérarchique. Il est composé de sous-processus ou activités. Les modèles de processus actuels utilisent des architectures de processus qui modélisent ces processus de la même manière. Cependant, la modularité de ces processus n'est pas le concept central pour bon nombre de ces approches. SPEM propose la notion de composant de processus, mais les interfaces sont limités à représenter des produits. La notion de processus est très proche au un service, où chaque composante de processus fournit un service si certaines conditions préalables sont remplies. Une encapsulation appropriée du processus devrait limiter les interactions, y compris les flux de contrôle et la chorégraphie à travers des interfaces spécifiées. MODAL améliore le concept de composant de processus offert par SPEM en utilisant la notion de ports qui offrent des services [Pillain 11]. EPC et WfMS n'offrent pas une approche spécifique pour l'encapsulation des processus.

L'absence d'une approche modulaire dans les méthodologies de modélisation des processus limite la réutilisation des processus modélisés. Une réutilisation opportuniste des processus dans une approche de modélisation de processus n'est pas une solution très élégante pour traiter la réutilisation. La réutilisation systématique des modèles de processus n'est possible que lorsque les processus sont conçus pour être réutilisés. De nombreuses applications de modélisation de processus actuelles, offrent la possibilité de stocker les processus dans des dépôts [Elias 12]. Ces solutions offrent des fonctionnalités différentes pour le stockage, la récupération et la gestion des versions. Le *Process mining* peut également être utilisé pour sélectionner le meilleur candidat pour la réutilisation [van der Aalst 07]. Mais trouver un procédé approprié pour la réutilisation est une partie de l'effort et intégrer dans un modèle de processus est une autre. Les approches de modélisation des processus n'offrent pas de moyens appropriés pour intégrer un processus réutilisables dans un modèle de processus. Une approche modulaire conçue pour la réutilisation peut aider à résoudre ces problèmes d'intégration.

Une méthodologie qui surmonte les lacunes des approches actuelles permettrait aux processus d'exploiter le raffinement pour le développement de processus par phase

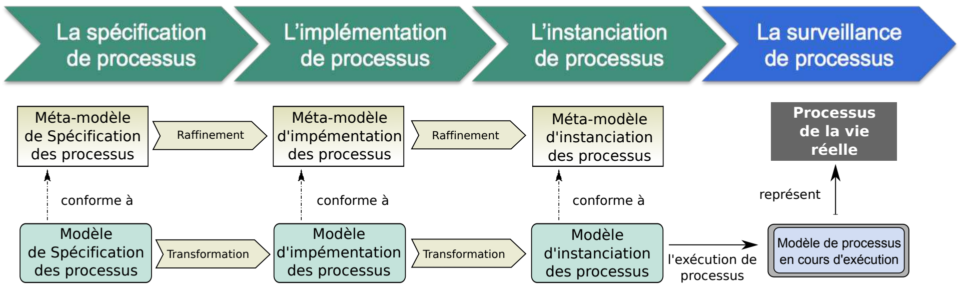

La spécification de processus
L'implémentation de processus
L'instanciation de processus
La surveillance de processus
Méta-modèle de Spécification des processus
Raffinement
Méta-modèle d'impémentation des processus
Raffinement
Méta-modèle d'instanciation des processus
Processus de la vie réelle
conforme à
conforme à
conforme à
représent
Modèle de Spécification des processus
Transformation
Modèle d'impémentation des processus
Transformation
Modèle d'instanciation des processus
l'exécution de processus
Modèle de processus en cours d'exécution

FIGURE 2 – Méta-modèles multiples pour la modélisation de processus

et la modélisation à plusieurs niveaux d'abstraction au sein d'une phase. Une telle approche permettrait de mieux comprendre le développement progressif des processus et favoriserait également la réutilisation des processus d'une manière plus systématique. Cette thèse présente un cadre de modélisation de processus et un outillage prototype associé qui répond à ces défauts.

2 Une approche multi-métamodèle pour la modélisation des processus

Pour un support de modélisation précis d'un processus dans une phase spécifique du développement, nous proposons d'utiliser un méta-modèle spécifique. Cela signifie que pour chaque phase du développement d'un processus, nous avons développé un méta-modèle distinct. Cela conduirait à une famille de méta-modèles de processus qui traitent la modélisation du même processus dans ses différentes phases de développement. Le nombre de méta-modèles dépend du cycle de vie de développement de processus choisi. Nous ne limitons pas l'utilisation d'un cycle de vie de développement de processus spécifique. L'idée de développement multi-métamodèle est basée sur une règle simple, « méta-modèle spécifique des processus pour la phase spécifique de développement de processus ». Pour illustrer notre approche, nous avons choisi quatre phases : la spécification, l'implémentation, l'instanciation et la surveillance de processus, comme le montre la figure 2. La phase de surveillance n'est pas une phase de développement, donc nous n'avons pas besoin d'un méta-modèle pour cette phase. Les utilisateurs de cette approche peuvent choisir d'autre phases et développer les méta-modèles nécessaires. Pour l'instant, nous présentons notre approche en utilisant trois méta-modèles, chacun pour la phase spécifique de développement de processus i.e. un méta-modèle de spécification des processus, un méta-modèle d'implémentation des processus et un méta-modèle d'instanciation des processus. Un modèle d'exécution est généré une fois que le modèle d'instanciation du processus est interprété par l'interpréteur de processus. Ceci permet d'exécuter les processus et, éventuellement, de les surveiller.

La figure 2 montre l'approche à deux niveaux de hiérarchie de modélisation. Dans la couche de méta-modélisation, nous voyons les trois méta-modèles et les relations de raffinement entre eux. Nous croyons que chaque phase du développement raffine le processus en ajoutant plus de détails. Dans la couche de modélisation, les modèles de processus respectifs sont illustrés en passant par une chaîne de transformation de modèles. Chaque transformation raffine le modèle de processus et ajoute de nouveaux créneaux dans le modèle, injecte plus de détails et des choix de conception.

Le modèle de spécification du processus est développé en conformité au méta-modèle de spécification du processus. Un tel modèle est basé sur la phase de spécification d'exigences de développement de processus. Un processus de développement logiciel est spécifié à l'aide de ce modèle, et donc n'est pas surchargé par des détails d'implémentation. Il peut être utilisé pour documenter les bonnes pratiques des processus en fonction de leur structure. Cela favorise la réutilisation des modèles de processus au niveau abstrait. Les standards de processus et les bonnes pratiques sont documentés de manière réutilisable, où ils peuvent être appliqués à n'importe quel projet ou d'une entreprise spécifique. Le modèle d'implémentation de processus est conforme au méta-modèle d'implémentation de processus. Ce modèle décrit les détails spécifiques à un projet, qui sont intégrés dans le modèle en ajoutant les détails d'implémentation du modèle de processus. La méta-modèle d'implémentation est sémantiquement plus riche pour exprimer les détails fins du modèle de processus dans la phase d'implémentation. De cette façon, un modèle de spécification de processus unique peut être utilisé pour plusieurs implémentations, dans différents projets sur plusieurs entreprises. Enfin, le modèle d'instanciation de processus est développé et établit les processus en les reliant à des outils de développement, des documents, des dépôts, des personnes, etc.

3 Les méta-modèles pour le développement du processus

Nous avons défini trois méta-modèles pour montrer l'applicabilité de l'approche sur un cycle de vie simple de développement de processus. Chaque méta-modèle est expliqué ci-dessous en fonction de sa pertinence pour la phase spécifique du cycle de vie de processus.

3.1 Le méta-modèle de spécification du processus

Le méta-modèle de spécification du processus est utilisé pour définir la structure de base du modèle de processus à l'étape de spécification. Un processus à l'étape de spécification est décomposé en différents composants de processus, ce qui crée une hiérarchie. Nous définissons un processus comme « une architecture d'activités interconnectés tels qu'elles visent collectivement à atteindre un objectif commun ». Une activité est une unité d'action dans un modèle de processus. Les activités peuvent soit être décomposées encore ou peuvent représenter le niveau primitif. Pour une hiérarchie de processus significatif dans CPMF, chaque activité composite contient un processus, ce qui revient à contenir une collection d'activités interconnectées. En plus d'offrir une hiérarchie de processus simple, CPMF permet le partage d'activité entre différents processus. Une activité peut être contenue dans deux processus différents, qui partagent certaines actions communes de traitement.

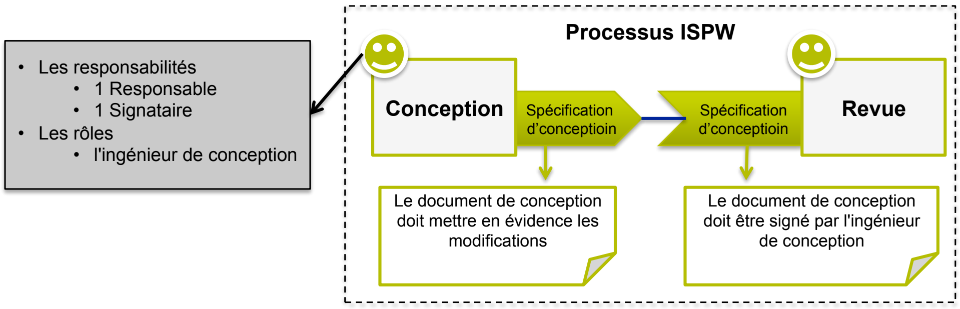

Les responsabilités
1 Responsable
1 Signataire
Les rôles
l'ingénieur de conception
Processus ISPW
Conception
Spécification d'conceptioin
Spécification d'conceptioin
Revue
Le document de conception doit mettre en évidence les modifications
Le document de conception doit être signé par l'ingénieur de conception

FIGURE 3 – L'exemple du modèle de spécification du processus

Inspiré de la conception par contrat (DbC), toutes les interactions de / vers le composant sont traitées par des interfaces spécifiées. Un contrat en CPMF peut être soit requis soit fourni. Un contrat est une spécification d'interface d'une activité pour les artefacts d'entrée / sortie. Chaque contrat d'une activité définit une spécification de l'artefact. Les contrats de deux activités sont reliés par des liaisons. Une liaison relie le contrat fourni d'une activité au contrat requis d'une autre activité. L'artefact fourni par la première activité doit remplir les pré-conditions de la deuxième activité. Ceci définit un flux d'activités sur la base de leurs artefacts. Chaque activité définit des responsabilités pour son traitement. Chaque responsabilité est attribuée à un rôle ou à une équipe. Le modèle de spécification de processus de CPMF peut être traduit de / vers les autres approches de modélisation de processus existants. Cependant, cette traduction provoque une perte conceptuelle.

Nous pouvons voir un exemple simple composé de deux activités *Conception* et *Revue* composent le processus ISPW dans la figure 3. *Conception* a un contrat fourni et *Revue* a un contrat requis. La spécification du document de conception est présent à la fois dans les contrats fourni et requis. Les rôles et responsabilités sont associés à chaque activité. *Conception* a un rôle associé d'ingénieur de conception avec deux responsabilités : signataire et responsable. Les interactions entre les activités sont représentées par l'utilisation des flots. Les post-conditions sont présentes dans le contrat fourni, par exemple *Conception* doit produire le document de conception avec la mise en évidence des modifications. Les pré-conditions sont présentes dans le contrat requis.

3.2 Le méta-modèle d'implémentation du processus

Le méta-modèle d'implémentation du processus est sémantiquement plus riche que le méta-modèle de spécification du processus. Il met l'accent sur la séparation des préoccupations, l'utilisation de mécanismes basés sur les événements et le dynamisme introduit par la modélisation multi-niveaux. Il y a deux hiérarchies parallèles définis par ce méta-modèle : une au niveau abstrait et l'autre au niveau concret. Au niveau abstrait, il y a une hiérarchie de processus abstrait qui contient les définitions des activités. Une activité de spécification méta-modèle devient une définition de l'activité dans ce modèle, *ActivityDefinition*. Cependant, il n'y a pas de spécialisation de

ActivityDefinition comme primitive ou composite. L'implémentation d'un ActivityDefinition est donnée par des *ActivityImplementations* au niveau concret. Il indique, si une *ActivityDefinition* est réalisée par la composition d'autres activités (composite ou primitive) ou est une *ActivityImplementation* primitive elle-même. Une *ActivityDefinition* se comporte comme un type d'activité (plus une conformité à une relation de type) et elle peut être réalisée par plusieurs *ActivityImplementations.* Ces implémentations de l'activité servent comme un ensemble d'implémentations alternatives pour la définition de l'activité. Chaque implémentation de l'activité comporte ses propres propriétés. Ces propriétés sont internes à une implémentation de l'activité et ne sont pas accessibles à l'extérieur, sauf à travers les contrats spécifiés.

Le contrat dans un modèle de processus est spécialisé par un contrat abstrait et un contrat concret. Chaque définition de l'activité spécifie un contrat abstrait, alors que chaque implémentation de l'activité spécifie un contrat concret. La relation *implements* entre l'implémentation de l'activité et la définition de l'activité est établie par une relation de conformité entre les contrats concret et les contrats abstraits. Un contrat abstrait d'une définition de l'activité porte principalement sur les artefacts, alors qu'un contrat concret traite des événements. Le contrat abstrait est spécialisé en trois contrats différents : le contrat d'artefact, le contrat de communication et le contrat de cycle de vie. Le contrat d'artefact présente la spécification d'artefact, qui est utilisé pour décrire les entrées et sorties de l'activité au niveau abstrait. En plus de préciser l'artefact, il présente également le méta-modèle de l'artefact. Le contrat de cycle de vie est un contrat abstrait qui définit un automate qui décrit le cycle de vie de l'activité. Le contact de communication définit les messages entre les rôles associés aux activités. Un événement de message dans le niveau concret est lié à ces messages, et est responsable de la chorégraphie réelle entre les activités / rôles. Tous les événements de contrôle appartenant au contrat concret sont liés aux artefacts spécifiés par la spécification d'artefact au niveau abstrait. Cette séparation des ressources contractuelles, permet la séparation du flux de données des activités de leur flux de contrôle. La définition du flux de données à un niveau abstrait (à part le contrôle de flux) permet de bénéficier de dépôts de données et de la gestion de la configuration. Alors que le flux de contrôle dans les activités utilisant des événements de contrats concrets peut être géré efficacement par le système de gestion des événements sous-jacent.

Le modèle d'implémentation du processus de l'exemple précédent est représenté sur la figure 4. *Conception* est une définition d'activité qui est placé au niveau abstrait. *Conception-Agile* est l'un de ses implémentations qui est placé au niveau concret. Donc on sépare les définitions de l'implémentation grâce à une architecture bicouches. Une seule définition de l'activité peut avoir plusieurs implémentations d'activité comme *Conception-Agile* et *Conception-RAD* pour *Conception* dans cet exemple. Le flot entre les définitions de l'activité est le flot de données du processus, comme entre *Conception* et *Revue.* Le flot entre implémentations de l'activité est le flot de contrôle du processus, comme entre *Conception-Agile* et *Revue-Stratégique.* Le flot de données utilise les spécifications d'artefacts, et le flot de contrôle utilise des événements pour l'interaction. Les rôles sont associés avec chaque implémentation de l'activité au niveau concret.

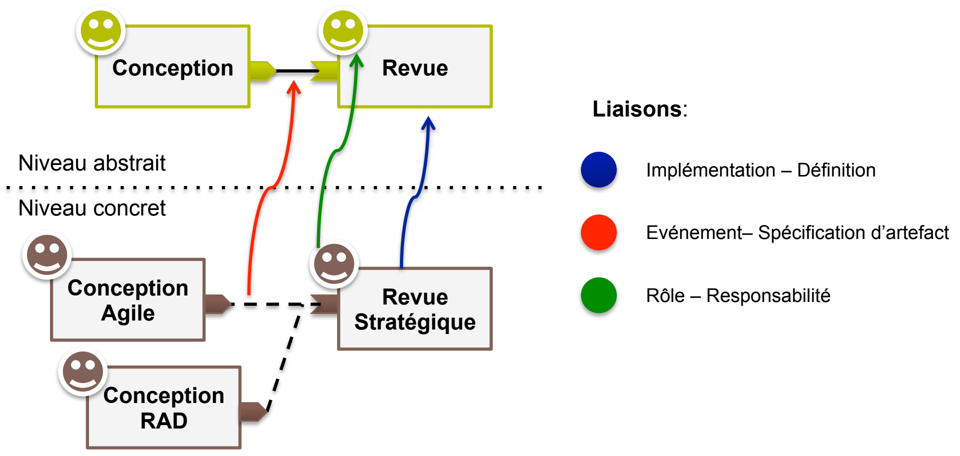


FIGURE 4 – L'exemple du modèle d'implémentation du processus

3.3 Le méta-modèle d'instanciation du processus

Les processus sont déjà conçus et implémentés avant la phase de l'instanciation. Le méta-modèle d'instanciation du processus se concentre sur la sémantique d'exécution du modèle de processus. Comme avec les méta-modèles précédents, la structure s'articule aussi autour de la notion d'abstraction. Cette abstraction permet à une activité de préciser sa dépendance sur les spécifications contractuelles des autres activités, plutôt que sur des instances concrètes spécifiques. Cette abstraction est exprimée à nouveau par une structure bicouche, où la couche supérieure est le niveau abstrait et la couche inférieure est le niveau concret. Un processus abstrait contient l'ensemble des définitions d'activité, où chaque définition de l'activité se comporte comme un type pour un ensemble d'activités d'instanciation. Une activité d'instanciation composite contient à la fois des activités d'instanciation et les définitions de l'activité. Une différence structurelle très importante entre le méta-modèle d'implémentation et celui d'instanciation est le contenu d'une activité composite de niveau concret. Une implémentation de l'activité composite dans le méta-modèle d'implémentation ne contient que le processus concret avec les implémentations de l'activité qui sont liés aux définitions d'activité. Il ne contient pas les définitions d'activité. A l'opposé, une activité d'instanciation composite à partir du méta-modèle d'instanciation contient à la fois le processus abstrait et le processus concret, ce qui signifie qu'elle contient un modèle de processus complet. Ainsi aucune activité d'instanciation n'est liée à une définition de l'activité hors de son contexte. Cela permet à une activité d'instanciation d'être complète pour l'exécution.

En plus des détails d'implémentation, les activités d'instanciation définissent également les détails de l'instanciation de l'activité comme la durée, la date de commencement, l'état d'exécution actuel, les itérations et l'itération courante etc. Une activité d'instanciation définit les contrats concrets pour les interactions avec d'autres activités. Les spécifications d'artefacts dans les contrats abstraits sont liées à des artefacts. Ces artefacts sont conservés dans un dépôt d'artefacts. Un artefact peut être soit une

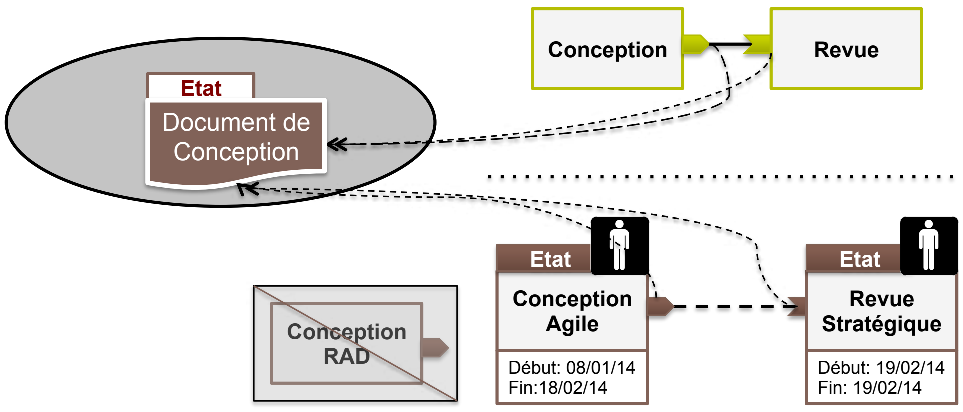


FIGURE 5 – L'exemple du modèle d'instanciation du processus

copie papier ou un document numérique. Chaque artefact numérique dispose d'un URL de dépôt unique, qui est utilisé pour l'accéder dans le référentiel d'artefacts. Le dépôt d'artefact supporte la gestion des versions, donc un artefact peut être verrouillé par une activité, en fonction de la nature de l'activité.

Nous continuons avec l'exemple précédent, présentant son modèle d'instanciation dans la figure 5. Les implémentations actives sont choisies pour chaque définition d'activité dans cette phase. Dans cet exemple, nous avons choisi d'utiliser *Conception-Agile* comme implémentation active pour l'exécution. Le flot de données lors de l'exécution entre les activités veux dire le transfert de l'artefact réel entre eux. Un dépôt d'artefact est utilisé pour le transfert d'artefacts entre les activités. Donc, *Conception-Agile* peut produire le document de conception et le placer dans le dépôt. *Revue-Stratégique* peut ensuite y accéder depuis le dépôt. Les états sont ajoutés aux activités et aux artefacts en fonction de leur cycle de vie. Les propriétés pour la planification et l'organisation temporelle sont ajoutées au processus pour l'exécution. Par exemple, les dates de début et de fin pour chaque activité. Enfin, les acteurs sont ajoutés aux implémentations d'activité qui jouent les rôles spécifiques. Par exemple, Fabien peut jouer le rôle d'ingénieur de conception pour *Conception-Agile*.

4 L'implémentation de l'outil de prototype

Un des problèmes avec les approches de modélisation de processus existants, c'est qu'ils ont une couverture limitée du cycle de vie de développement de processus. Les concepteurs de processus ont besoin d'une approche pour la spécification de processus, de les transformer en des approches d'implémentation et de les transformer encore en des approches d'instanciation pour rendre le modèle de processus exécutable. Certaines de ces approches couvrent à la fois les phases d'implémentation et instanciation, mais une transformation est nécessaire pour une couverture complète du cycle de vie du développement de processus. Ces transformations d'une approche à une autre provoque des pertes sémantiques, en raison de plates-formes de modélisation de processus incohérentes. CPMF fournit un prototype qui peut être utilisé pour développer des modèles de processus pour le niveau de spécification. Ces modèles de processus sont développés en utilisant un éditeur de processus graphique fourni avec le prototype ou à travers un langage de domaine spécifique textuel. Les modèles de spécification d'un ou plusieurs processus sont ensuite transformés en des modèles de processus d'implémentation en utilisant un moteur de transformation, fourni avec le prototype. Ce moteur de transformation transforme les spécifications de processus en des implémentations et de ces implémentations à des instanciations.

Une fois que les modèles de processus sont raffinés pour la phase d'instanciation, ils deviennent exécutables. Ces modèles de processus exécutables peuvent être chargés dans l'interpréteur de procédé fourni dans le prototype. L'interpréteur de processus développe les instances de modèle de processus et les exécute. Une interface web permet d'interagir avec les processus en cours d'exécution. Cette interface web est appelée tableau de bord de gestion de projet. Le propriétaire d'un processus peut accéder à toutes les activités d'un processus qu'il / elle possède. Les acteurs jouant des rôles spécifiques liés aux activités peuvent accéder à ces activités seulement. Une fois que les artefacts requis par une activité sont disponibles, l'acteur associé peut télécharger les artefacts requis, faire le traitement et télé-verser les artefacts développés à travers cette interface web. Le propriétaire d'un processus peut adapter les activités en changeant son implémentation active en une des implémentations alternatives déjà développées dans la phase d'implémentation. L'adaptation des activités dans un modèle de processus d'exécution doit s'occuper du transfert de l'état entre l'ancienne et la nouvelle implémentation de l'activité. Ceci est géré par l'interpréteur en utilisant des liens en dur, soit par les constructeurs d'implémentation d'activités ou en présentant les propriétés des deux implémentations au propriétaire du processus qui peut réaliser le transfert de l'état.

5 La méthode de développement de processus

Les modèles de processus de spécification sont utilisés pour développer des standards de processus. Plusieurs standards peuvent être développées dans cette phase. Ils peuvent être stockés dans le dépôt pour une utilisation future. Ces standards sont ensuite transformés en modèles de processus de l'implémentation. L'implémentation des modèles de processus assure la conception détaillée de chaque activité. Plusieurs implémentations d'une activité aide pour le développement d'une base de connaissances des implémentations. Plusieurs modèles de processus de spécification peuvent être utilisés pour développer un seul modèle d'implémentation. Ceci est utile pour les cas où la conformité à des multiples normes est nécessaire.

Les modèles d'instanciation sont utilisés pour ajouter des détails spécifiques du projet. Ces modèles de processus sont exécutables. Les liens de traçabilité entre les trois niveaux permettent de remonter jusqu'aux spécifications. Finalement, le modèle de processus est exécuté. L'interaction avec le modèle de processus permet la gestion des processus. Leur état peut être contrôlé. Les modèles de processus peuvent être adaptés à l'exécution.

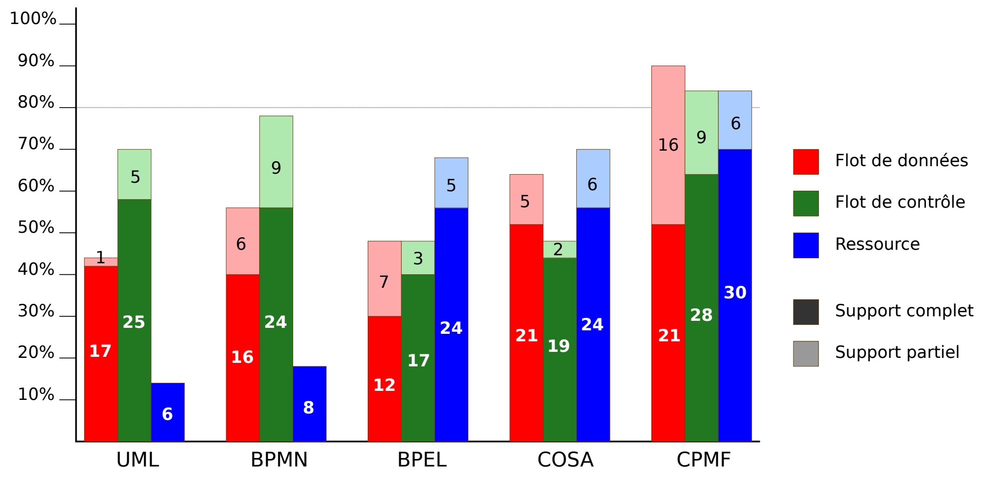


FIGURE 6 – Évaluation basée sur *Workflow Patterns*

6 L'évaluation de l'approche

Pour valider notre proposition de recherche, nous avons utilisé un cas d'étude et une évaluation basée sur des *Workflow Patterns*. Le cas d'étude porte sur une entreprise fictive, *TB-Enterprise*. Il présente l'application de CPMF sur un scénario pseudo-réel pour élaborer et justifier l'hypothèse faite dans cette thèse. Le scénario choisi pour cette étude de cas est pseudo-réel car le processus original est modifié pour illustrer les concepts clés de cette approche. *TB-entreprise* développe le processus de test de *AlphaSystem* par un ensemble de modèles, correspondant chacun à une phase spécifique du cycle de vie de développement de processus. Il montre comment ces modèles sont raffinés. Il a également porté sur le conformité à plusieurs standards de processus. Ce cas d'étude démontre également la façon dont les acteurs interagissent avec l'exécution des modèles de processus à travers le tableau de bord de gestion de projet. Comment leurs droits d'accès à certaines activités et certaines actions liées à une activité spécifique sont gérés par le tableau de bord.

Il existe trois axes principaux d'un modèle de processus, son flot de contrôle, son flot de données et ses ressources. Le *Workflow Patterns Initiative* présente une collection de patterns pour tous ces axes. Les résultats de l'implémentation d'autres approches sont fournis par l'initiative pour l'étude comparative. Nous avons mis en place ces *patterns* dans notre méthode et utilisé les résultats pour la comparaison avec d'autres approches. Les résultats sont illustrés par le graphique de la figure 6. Nos résultats sont comparés avec les approches traditionnelles en utilisant des diagrammes UML d'activité, les approches de *business process management*, BPMN et BPEL et un approche de *workflow management*, COSA. Les résultats de la comparaison avec les autres approches sont très encourageants dans les trois axes. Nous sommes en mesure de supporter complètement et partiellement plus de 80% des *patterns* dans les trois catégories. Les *patterns* qui sont pas pris en charge par notre approche ne le sont pas en raison de défauts de la méthode. Les raisons de l'absence de support sont le choix de l'implémentation ou les limitations du prototype. Par exemple, l'un des *workflow* *data patterns* requiert le transfert d'artefacts par valeur entre les activités. Nous avons choisi d'implémenter l'interpréteur en utilisant le transfert par référence et un dépôt d'artefacts. Cela ne peut pas être considéré comme un défaut de la méthode.

Chapter 1

Introduction

Contents

1.1 Context

Software Engineering, a term coined by Professor Fritz Bauer in 1969 [Naur 69], is an approach that encompasses all the activities of design, development and maintenance of software, and the systematic and organized procedures to streamline these activities. This application of *engineering* to software domain is focused on dealing with growing software complexity and scaling the solution to the problems in a quantifiable fashion. Until 1980's the classical lifecycle process models were dealing fine with the development of software systems. But the advances in software technology made it evident that those traditional methods of conducting the software development process were not adequate enough to deal with the growing complexity of the software systems. Under the umbrella of lifecycle models, the core methods to deal with the control of development processes were ad-hoc in nature. With increasing expectations of more features, those ad-hoc methods were becoming incapable to keep up with the growing need of quantity and quality of software. In order to respond to this demand, a consistent methodology of software development process was required.

When the quality of these methods was put into question, the research community responded with some process centric approaches. This led to the emergence of software process modeling methodologies that focus on the definition and analysis of the software development processes. The first step towards a coherent and consistent methodology to integrate individual activities contributing to a larger goal was targeted towards their coordination. An initial coordination language was proposed as soon as 1983, which was specifically targeted towards the software development activities [Holt 83]. This coordination language evolved in Diplans [Holt 88] and then further into the Role Activity Diagram (RAD) notation [Ould 95]. Even though, this notation did not meet the needs of software industry, it was highly welcomed by the more general business process community for the description of organizational processes. Another important notation, UML Sequence Diagram [Booch 97] was also proposed at that time to model organization processes. This notation focused on the description of interactions between objects within a system through a set of message exchanges to achieve the desired result [Li 99]. Gradually, this notation was also used to represent the interactions between business processes and actors, and thus served as a basic process modeling notation for quite some time [Hruby 98].

Around the same time when these graphic-based process notations were evolving, Osterweil introduced a notion of a process program in his key paper published in 1987 [Osterweil 87]. He described the nature of a process as "a systematic approach to the creation of a product or the accomplishment of some task". He emphasized on their systematic nature argued that they should be dealt with in the same manner as other software, and highlighted the concept of process programs. This gave birth to the term *process modeling language*, which was software programming like language composed of process objects created by a development process. These objects are themselves the description of the process. This enabled the execution of these process programs on computers and thus allowed them to interact with the real-life process that they model, while being *enacted*. This allowed an enacted process (executing) to be in a suggestive or controlling position with respect to the real-life process that it models. Shortly after this, Process Support System (PSS) [Bruynooghe 91] was introduced by the Alvey APSE 2.5 Project [Snowdon 90]. An important part of Process Support System was the Process Modeling Language (PML), which modeled process as a network of loosely coupled asynchronous sub-processes.

As the importance of modeling a process and benefiting from the amount of automation that it can provide was coming to light, more approaches were proposed both by academia and industry [Chung 89, Curtis 92]. As described earlier these approaches could be categorized into two distinct categories: text-based and graphic-based. These techniques are now mostly referred as process programs and process modeling notations. Process programs at that time were offering the added advantage of execution. Apart from enacting the process, these execution mechanisms allowed to build tools that could analyze these processes and their effectiveness while being enacted and possibly in real-time. On the other hand, execution of the process modeling notation was not possible until the recent developments of the Model Driven Engineering.

All forms of engineering rely on models in order to understand the complexities of the real world systems. They allow an engineer to abstract out the non-necessary details of a physical system in order to focus and reason about the relevant concepts. They can be developed before developing the physical system or can be derived from the existing system to understand it [Selic 03]. Models are used for different reasons: predicting the properties of the system, reasoning about its behavior, communicating across different technical spaces, etc. But in software domain, they are currently being used to automate the development of the software system [Kent 02]. Different aspects of a system can be considered relevant at different points in time, and each of these aspects can be modeled (as a perspective) using different model elements. In such cases some hints or rules can augment these models in order to transform them to the other perspective. These rules can also be used to enrich these models with more information in order to refine them. These models and model transformations form the core of the Model Driven Engineering (MDE). Recent advancements in MDE offer the tooling support that can be used by system architects and developers to model the system perspectives and maps them directly to the programming language code for a particular platform [Jouault 06].

Software processes just like software systems are based on the notion of lifecycle, where each stage of development has different concepts to frame. Each phase of a software process organizes different factors and issues related to the degree of its maturity in terms of completeness. Fuggetta defined a software process as, *the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product* [Fuggetta 00]. But we tend to think more like the Osterweil's analogy of a software process, where he advocates that processes share the same nature and complexity as the software system and should be treated the same way [Osterweil 87]. Thus we are of the view that software processes themselves need to be conceived (specification), developed (implementation), deployed (enactment) and maintained (monitoring). As the lifecycle phase of a software process advances, the focus on issues related to the process also changes. For example, in specification phase the focus remains on *which* artifacts would be handed over by the activity. On the other hand, for enacting a process the focus shifts to *when* should an artifact be handed over by this activity. During the enactment phase, the reasoning about the activity is not targeted towards the choice of its inputs and outputs, rather its directed towards the related *hows* and *whens*.

The current process modeling languages appear to have two distinctive problems. First, they seem to ignore the importance of a consistent approach that handles process in all the stages of its lifecycle. Either a single process model seems to represent a process in all phases (*e.g.* at specification phase and the implementation phase) or it has to be transformed to a completely different approach for enactment. For example BPMN [OMG 11] models processes in all stages of development through a single notation, however it does not offer the possibility to enact them. Consequently, process developers are bound to transform the models to BPEL [OASIS 07] for enactment. Second, most of the approaches focus on the flow of activities defining the order of (presumable) execution. Some approaches use event based mechanisms to induce reactivity, but still their focus remains on the flow of activities [Adams 05]. Approaches like Event-driven Process Chains (EPCs) use both types of inputs for the activities (events and artifacts) together, which clutters the process model [van der Aalst 99]. Dataflow and control flow in a process, at the same time, makes it hard to conceptualize the interactions of an activity to its context.

A methodology that overcomes these shortcomings of the current approaches would allow the processes to exploit refinement in terms of phase-wise process development and multi-abstraction modeling within a phase. Such an approach would make it easier to understand the development of processes over time and would also promote reusability of processes in a more systematic manner. This thesis presents a process modeling framework and associated tool support that responds to these shortcomings.

1.2 Problem Statement and Research Questions

The following scenario illustrates the problems to be addressed in this thesis. An aerospace company needs to develop a mission critical software for their new project under a tight deadline. This company would like to reuse the past experiences of developing software and plans to comply with some organizational and international standards. Their past business process knowledge is documented in the form of process models, implemented in the current approaches. Their organizational standard is also well documented and this time they have to follow the ECSS standard as well. They are looking forward to the possibility of following NASA standard along with the ECSS standard, so that their software can be sold to a greater market beyond Europe. One of the key considerations is to be able to evolve their process models to follow the iterative approaches and support process improvement. When an enterprise has to deal with mission critical software, it has to make sure that it has contingency plans to reduce the risks. These contingency plans are the redundant processes that might be used if the things do not advance as expected. This means that the running process can be adapted anytime, using these alternative contingency processes. Last but not least, it is highly desired to be able to reuse the processes that they develop. The following problems may arise in this scenario:

— **Reused model contains a lot of noise:** The processes that this company wants to reuse are detailed with a lot of implementation noise, which makes it difficult to separate the conceptual and the implementation details. For example, the previous process model was built in BPMN [OMG 11] and enacted in BPEL [OASIS 07]. From BPMN, it seems very difficult to find out that a sub-process was itself part of a standard or it was an implementation of the super-process to achieve the goals as described in the standard. Besides this, the demarcation between requirements and design becomes confusing. For example the notion of *state* concerns the implementation and enactment phase of the processes, it is a noise for the requirements phase. This enterprise would encounter same problems if the past process models were built using SPEM, Activity Diagrams or Workflows.

— **Conceptual loss across platforms:** Let us assume that past processes were enacted in BPEL. Because of the transformations and/or manual development from BPMN, many of the conceptual details are worked around to comply with BPEL. For example end events are transformed into invoke activities, throw activities or reply activities. And the type of activity depends on the output artifact type; if there is no output, no activity is generated in BPEL. Swimlanes are not mapped. Sub-processes are also mapped to invoke activity. (See the mapping provided in the BPMN specification [OMG 11]). And a much bigger problem is lack of standard transformations, thus every tool can produce a different result. The same problems occur if the original process model was built in SPEM [OMG 08]. Lack of standardized transformations between implementation and instantiation processes alone can jeopardize the overall standardization of mission critical system processes.

— **Insufficient support for adaptation:** In this scenario we want the process to be evolving, which means that the activities can be replaced, amended or improved frequently. If the process model is developed using BPMN, SPEM, Workflow nets [van der Aalst 97] or any other approach that focused on the static flow of activities [Chang 01], one needs to re-plan the flow, each time a process optimization in terms of flow is suggested. As the focus remains on flow so every activity expects some data/event from a precedent activity and gives out to the subsequent activity. It means that dependency of an activity is expressed on other activities rather than the required artifacts. If an activity is replaced with another, the new activity needs to be reconnected with the pre and post flows. The focus of an activity should remain on the inputs and outputs, in a way that if the input of an activity (no matter where from it comes) is legitimate, the activity should work.

— **Compliance with standards not assured:** When activities in a process model are expected to replace or amend frequently, it is highly recommended to ensure that somehow this change does not breach the standards being followed. This is particularly important when the replacement activities are developed at runtime. Current techniques like BPMN, SPEM, Worklets [Adams 06], Yawl [van der Aalst 05c], do not allow any mechanism to ensure this compatibility of the process model to the adopted standard.

— **Loss of integrity:** Because the focus of the process does not remain on the input and output contracts of the activities, it is not guaranteed that two old processes despite have resembling provided workproducts and required workproducts could be assembled.

— **Inefficient support for teamwork:** With new software development lifecyle models like XP and pair programming, it is not possible to constrain that one workproduct can only be a responsibility of one role. With growing notions of teamwork, it is not possible in BPMN [OMG 11] to share things in swimlanes or even have multiple *responsibles* for a workproduct in SPEM [OMG 08].

— **Lack of process and development environment integration:** With the current advancements in the domain of modeling, this company would like to use a fair amount of automation that can facilitate the development of software process models. Apart from facilitating the development of processes, the integration between the software development tools and the software process modeling tools could be used to automate different activities, while the enactment of the process model. Many of the current process modeling frameworks do not allow this level of support. A very recent initiative of WfMC (Workflow Management Coalition) presented a tool support for WfMC reference model [Hollingsworth 95, Hollingsworth 04]. This tool, Stardust [Eclipse 13] has recently offered this possibility for the users of eclipse.

— **Multi-standard compliance not supported:** As stated earlier, this company has some well documented organizational standards that they follow. Thus the process being developed will have to comply with this internal standard. But the problem arises when they are looking forward to comply with two other standards at the same time along with this internal standard. They are planning to develop the process in a way that their development procedures should be considered standard beyond Europe, so they are planning to follow ECSS and NASA both the standards. Now the current approaches either do not support compliance with standards and if they do they do not support compliance with multiple standards. By the lack of support for compliance with standard, we do not say that it is not possible to do that, it is just that manual workarounds are used in practice to achieve that. The technologies themselves give no support in this regard.

— **Dynamic process creation/destruction not supported:** Many of the popular approaches used today are not very dynamic in nature, where the process models show little or no reactivity. Even the process models that are considered dynamic, either propose alternative process routing or support exception handling. Both these approaches are integral to a well developed process modeling approach, but they do not exploit process dynamics to its limits. For a mission critical system a predefined process compensation can help in many situations, but it is desirable to be able to manipulate the process in a more dynamic way, where a process can create and destroy new subprocesses at runtime. This can be achieved through the use of process factories, that are accessible to the running processes.

— **Lack of structure for artifacts:** Let us assume that the process under development is itself guided by the input artifacts that it takes along the execution. For example our process has an activity that takes an automatically generated report as its input. Based on the contents of that report, the execution path of process is defined. Many of the current approaches allow the possibility to receive *accepted* or *declined* events or to have preconditions based on the properties of the input artifacts. When it comes to taking the decisions based on the contents of the artifact, manual activities (where humans are the performers) can easily handle such situations. But for automatic activities to be able to read the contents of the inputs, it is highly desirable that the artifacts are themselves taken as models that conform to their respective metamodels.

In order to cope with the problems related to separation of concerns regarding to different phases of process development, a refinement-based approach is proposed, where the details should be added to the process incrementally. Component oriented paradigm provides a good inspiration to study the impacts of contractual interactions between the processes to eliminate the focus on dataflow only. It can further help in issues related to process manipulations where componentization can ensure process integrity. This inspiration also takes advantage of the refinements where component types, component implementations and component instances serve as an example to guide the processes in the same fashion. The key to solving these problems is to look into the depths of refinement-based approach and benefit from what it offers.

Hence the main research question that the thesis attempts to answer is: How can a process modeling approach be optimized for automation using constructs that are refined for each specific phase of process development? More specifically,

— **RQ-1:** In an approach where activities are defined by their contracts, how can software development processes benefit from the contractual interactions between the activities?

— **RQ-2:** How can process reuse be fostered, while focusing on the process development lifecycle?

— **RQ-3:** Some techniques rely on data-flow while others on control-flow. What benefits could be achieved by merging them at different levels of abstraction?

— **RQ-4:** How can processes benefit from compliance of standards and further from multiple standards?

— **RQ-5:** If a standard methodology is used to model the process from specification to implementation till enactment offering backward traceability. What would be the gains?

— **RQ-6:** How can automation be exploited for the development of processes and then used by these processes to develop the artifacts?

1.3 Solution Criteria

In order to ensure that we have a clear definition of the qualities that a suitable solution to the problems identified in the preceding section should demonstrate, we nominate the following criteria as a means of assessment.

1.3.1 Completeness

The definition of processes where the levels of abstraction are refined *fully* is referred as complete or fit for enactment [Feiler 93]. Boehm suggests that a good approach to verify the completeness of a process is to follow the WWWWWHH principle [Boehm 96]. Following questions need to be answered in order to follow this principle:

— **Why:** Why is the system being developed? (objectives)

— **What/When:** What will be done and when? (milestones and schedules)

— **Who/Where:** Who is going to do and where? (responsibility and organizational structure)

— **How:** How would the activities be carried out (in terms of technique and management)? (approach)

— **How much:** How much of the resources would be needed to carry out the process? (resource)

However the completeness of a process definition depends on the context, e.g. it may be complete for one stakeholder and may not be, for another. But the completeness of a process modeling approach should allow all necessary constructs, at the right time, to synthesize the process. We believe that a process modeling approach should be able to allow the refinement of abstract constructs to a level that the system can be modeled without any loss of information. This is the prime aspect that we have focused in this thesis, while presenting our process modeling framework. An important aspect regarding the completeness of a process modeling approach is to offer the constructs that show the intent for each activity. Such constructs can be used to analyze the objectivity of the processes with respect to the business goals [Cortes-Cornax 12]. Finally, a process modeling approach should be considered complete, if it defines the process behavior. Many approaches do not offer formal semantics, but if they specify the behavior even informally, we consider it to be complete.

1.3.2 Team Development

A process model that supports team development needs to go beyond simple allocation of tasks. A comprehensive methodology that enables the process designer to define the privileges of each role in a specific activity needs to be incorporated. Responsibility assignment matrices are one of the approaches that can be integrated within the process models for the detailed definition of the privileges associated with each role. In order to support team development, a process modeling approach needs to provide some support for team communication as well. It serves for the exchange of messages between the actors associated to the activities.

In order to support team development, a process modeling framework should allow distributed access of the process models. The process components can be assembled by a different person than the one who developed them and it may happen much long after they were built. For this to work seamlessly, activities should be separated and developed with clear dependencies, and their interfaces should be specified unambiguously. The architectural conventions and rules must be explicitly specified. One of the benefits of using contract based process architecture is the explicit specification of interfaces. This allows the process designer to assemble the activities developed from different process developers, even from different organizations. As contract based architecture for process modeling promotes the decoupling of activities, their development can be easily managed for distributed teams.

1.3.3 Reusability

Reuse is the usage of previously developed or acquired concepts and objects in a new context, which may be classified as the reuse of knowledge and the reuse of artifacts and components [Prieto-Diaz 87]. It can be of two types: *opportunistic*, where the design was not made to support reuse and *systematic*, where the system is designed for reuse. During the development of a process model, process components are the artifacts being produced. These components should be developed in a manner that supports systematic reuse. Contract based architecture for process modeling allows these components to have specified interfaces and dependencies [Crnkovic 06]. This means that a process component can be a feasible candidate of a process requirement in a different context, if it complies with those requirement specifications.

The goal of reusability is to minimize the effort of re-development of a component and promote standardization in an enterprise. In order to support reusability in an efficient way, we believe that adaptations of a required process component to fit in the new context should be minimized. These adaptations can be minimized if a process component can be acquired from a variety of process development phases. For example, a *test* activity component can be reused either from specification phase or implementation phase, as per requirement. This promotes the reuse of process component at a desired level of abstraction. Use of refinement based approach for process development allows developing a process in multiple phases, where each phase corresponds to a different level of abstraction. We consider reusability as a solution criterion, that can be achieved in process modeling approaches in ways: 1) approach based, where the core process modeling approach is design for reuse 2)implementation based, where the implementation tools for the approach allow the reuse of process fragments, despite a lack of support from the core approach. Implementation based reuse is less efficient because reuse across multiple implementation tools become difficult.

1.3.4 Abstraction

Any approach that uses modeling is fundamentally based on abstraction, as models are themselves an abstraction of the concerned system. However, the use of abstraction does not end by modeling a system. It can be further exploited to develop the models in multiple levels. Each high level model, in this case, is a smaller model obtained by abstracting away some information from the lower level models. A process modeling approach can use this kind of abstraction to develop multiple models, where each low level model is a refinement of a high level process model. Abstraction can also be exploited by the use of typing mechanisms in a process modeling approach. Typing mechanisms allow to develop different types of processes, which can be specialized within the same process model. This allows to structure the process models, based on the concepts commonly used in object oriented paradigm. A process model itself can be built using multiple abstraction layers having conformance relationship between them. In such scenarios, each layer is a refinement of the higher level layer.

1.3.5 Modularity

A logical partitioning of a complex system into smaller parts that are easy to implement, manage and maintain is the key idea behind modularity. We consider two kinds of modularity for the purpose of this thesis: hierarchical and contextual. Hierarchical modularity refers to the tree like structure of a system, where one node *contains* other nodes. Processes are hierarchical by nature. A process might be made up of some sub-processes which might in turn be composed of other sub-processes. This composition of processes continues till a very fine level, where they can not be decomposed any further. Such processes are normally termed as tasks, primitive activities, *etc.*

Contextual modularity is the partitioning of a parent process into sub-processes, but the focus is on multiple sub-processes that can be composed together to develop the parent process. This means how effectively, a parent process is decomposed into sub-processes. What design is chosen for the interaction of sub-processes, such that they can collectively form the parent process. Each sub-processes should define some interfaces so as to interact with other sub-processes in the same context. Definition of interfaces allow contractual interactions between the sub-processes.

1.3.6 Tailorability

For the reuse of process components, ideally, the previously developed process should match with the requirements of the new context and it should fit in as it is. But we assume that the available activities do not completely match the requirements of the new situation perfectly; this brings us to the problem of adaptation, more precisely process tailoring. There are scenarios, other than process reuse, where process tailoring can be of high interest as well, e.g. process improvement [Johnson 99, Hurtado 12].

Process tailoring is a characteristic that can be provided by the process modeling methodology if it is flexible enough to support process changes. But allowing for process tailoring is not the only issue to be considered when a standard has to be followed. In such cases, one has to make sure that the new changes still comply with the standard [Sadiq 07]. A mechanism that allows enough flexibility to change the process without interfering with the core process objectives should ensure that process offers the standard interfaces required from it. A certain level of flexibility can be provided by the use of process tailoring, when the process framework provides multiple variants of the process. All these variants are pre-tailored to different implementations, where the feasible candidates for enacting the process can be chosen, based on the runtime context. Runtime adaptations to executing process models is also an important aspect of process tailoring. Having multiple variants allows to replace them on the fly.

1.3.7 Enactability

The first solution criterion described in this section was the completeness of processes, which is fundamental to carry out the routine tasks for process automation. If a process has a sufficient level of precision, many of the routine tasks can be automated through the use of process enactment tools and their integration with the software development tools. Software engineering being a creative science has many processes that can not be automated completely. However for manual activities, a certain level of automation can be achieved to help the actors in designing, planning, controlling and monitoring the process.

A process instance should have all the required elements of implementation details to make it enactable. These implementation details consist of the required inputs for the process, the assigned roles (specially the initiating role), resources, initial enactable state and continuation and termination capabilities [Feiler 93]. A process that does not have any of these elements can not be enacted. Once a process is enacted, it is capable of interacting with the real life process that it is modeling. Processes vary in terms of their capabilities to be automated, where they can be manual, semi-automatic or automatic. An enacted process may take a suggestive position for manual and semi-automatic processes. It can be in a controlling position for semi-automatic and automatic real life processes, even though a human role is always necessary as a responsible role.

1.4 Approach

In the light of the current context in process modeling, its difficulties and limitations, we propose a method that guides organizations to perform a phase-wise software process modeling where each phase refines the previous one. In order to achieve this vision, four distinct research activities are undertaken.

1.4.1 Phase-wise identification of the core process constructs

In order to develop a comprehensive approach for software process modeling that covers each phase of process development, it is first necessary to identify the core constructs of the processes at each phase. We believe that a process modeling approach that is based on refinement, should be able to model the processes in different phases on a need basis. A process model of a specific phase should not be polluted with the constructs that are not used in the current phase. We do not enforce a process lifecycle to have only three phases of development, but for the sake of explanation, we have chosen to remain within three phases *i.e.* specification, implementation, instantiation. For each of these phases, an empirical survey of existing approaches, modeling formalisms and standards is undertaken, so as to enable the identification of generic constructs used in them. As we compare these constructs to the existing approaches that do not place them according to the lifecycle phases, we reason about their placement within different phases and provide examples and motivations for it. [Research questions addressed: RQ-1, RQ-2]

1.4.2 Synthesis of process constructs into respective metamodels

The generic constructs of a process model identified in the previous research activity provides a basis to model a complete process in each phase of development. Based on these constructs a metamodel (process modeling language) has been developed for each phase of process lifecycle. Each process metamodel in this framework has sufficient details about a process to carry out the tasks related to that phase. As the process lifecycle advances, it is modeled using a different metamodel that refines it further with more details pertaining to the current development stage. The broad range of concepts embodied by the process framework are injected in process model on a need basis, as it gets refined over time. As a process model advances from its specification phase to implementation phase, it is transformed into a bi-layered model, where process definitions are separated from their implementations. Both these layers co-exist in the subsequent model(s), where one guides the sequence of activities based on data-flow and the other on control-flow. To provide unambiguous interpretations for each of these bi-layered metamodels, the motivations are discussed alongside a running example. [Research questions addressed: RQ-2, RQ-3, RQ-4]

1.4.3 Development of the tooling support

As the process modeling approach presented in this thesis deals with the modeling of software development processes that can be modeled till the enactment phase, its capability can not be guaranteed unless a platform is provided to enact the processes. As we model processes from the specification till enactment, so the first tooling support comes from the designing phase, where a model editor is developed to model the specification models. The model editor can be used to develop the subsequent phase process models as well. This process editor is based on Openflexo [Openflexo 13], an open source collaborative modeling framework that allows to generate model editors based on viewpoints. A process interpreter is developed to enact the process models. Once the process is being enacted, a web interface allows to interact with the executing activities. The enacted process is controlled and monitored through an administrative panel on the web interface. Artifact repositories are created where each artifact is versioned and is made available to other activities based on their access rights. Process repositories are created to store the process knowledge that can be reused to develop new processes. [Research questions addressed: RQ-5, RQ-6]

1.4.4 Evaluation of the process modeling framework

In order to explore the applicability of the proposed process modeling framework and to uncover improvement opportunities for the methodology, we evaluate this thesis in two manners. Initially, a pseudo-realistic process model is used to validate the proposed process modeling framework. The implementation and enactment of this model is used to reason about the syntax and semantics of the approach. Finally, the approach is evaluated against the established workflow patterns, so as to place our framework in the existing state of the art. Workflow control-flow patterns [Russell 06a, van der Aalst 03b], data patterns [Russell 05a] and resource patterns [Russell 05b] are implemented by the proposed framework so as to identify the strengths and weaknesses of the framework. The aim of this evaluation phase was twofold. First, to demonstrate that the proposed approach could successfully enact a process model which covers the software development processes of an enterprise. Second, to identify loopholes in the proposed framework, that should be worked upon to optimize this approach.

1.5 Scope & Contributions of this Thesis

The problems (identified in the beginning of this chapter) associated with current process modeling approaches affect their applicability under varying circumstances. The process modeling approaches are either too generic and are normally targeting the business world, or are focused to the procedural style of workflow. On the other hand Software process languages are very complicated and are difficult to understand for the process designers with minimal backgrounds from programming domain. Furthermore different process modeling approaches target a specific designing or execution phase of process development. A consistent approach that can handle every phase of process development is long past due. The overall scope of this thesis is to fill this gap.

The main contributions of this research work are:

— **FUNDAMENTAL:** To develop a set of process metamodels (three metamodels for demonstration purpose only) that can define languages to model the software development processes in different process lifecycle phases.

— **FUNDAMENTAL:** To foster a consistent refinement approach that keeps on enriching the process models in parallel with the advancing process lifecycle phases. With a consistent approach, we mean that no transformations are needed between two different languages e.g. BPMN to BPEL for execution.

— **FUNDAMENTAL:** To present a bi-layered methodology of modeling software processes, where data-flow and control-flow are separated. A mapping between events and artifacts ensures the coherence between the two levels of flow.

— **FUNDAMENTAL:** To demonstrate that a bi-layered refinement based approach can be exploited to comply with single or multiple process standards, throughout the process lifecycle. Backward traceability allows to reason about each activity and relates it to the standard being followed even in previous lifecycle phases.

— **EXPERIMENTAL:** To develop a prototype implementation of the proposed software process modeling framework that can be used to develop the process models and enact them. A project management dashboard to interact with the enacting processes is also presented.

- **EXPERIMENTAL:** To validate the process modeling methodology and prototype by developing a pseudo-realistic process model that covers different kinds of activities one can confront in a real life software development project.
- **EXPERIMENTAL:** To evaluate the process modeling framework against the established workflow patterns to place our framework in the right place amongst other related approaches.

1.6 List of Publications

- F. R. Golra, Fabien Dagnat *The Lazy Initialization Multilayered Modeling framework: NIER track.* In *Proceedings of the International Conference on Software and System Process (ICSE 2011).* ACM/IEEE, Hawaii, USA, 2011.
- F. R. Golra, Fabien Dagnat. Using component-oriented process models for multi-metamodel applications. In *Proceedings of the 9th International Conference on Frontiers of Information Technology (FIT'11).* IEEE, Islamabad, Pakistan, 2011.
- F. R. Golra, Fabien Dagnat. Generation of Dynamic Process Models for Multi-metamodel Applications. In *Proceedings of the International Conference on Software and System Process (ICSSP'12).* IEEE, Zurich, Switzerland, 2012.
- F. R. Golra, Fabien Dagnat. Specifying the Interaction Control Behavior of a Process Model using Hierarchical Petri Net. In *Proceedings of the 2nd Workshop on Process-based approaches for Model-Driven Engineering (PMDE 2012).* Copenhagen, Denmark, 2012.

1.7 Outline of the thesis

This thesis is organized in three parts. Part I explains the context of the approach and presents the state of the art in process modeling approaches and standards. Part II identifies the core constructs in different metamodels of the proposed process modeling approach. It also explains the implementation of our approach and discusses how the identified problems are resolved by using this approach. Part III evaluates our approach by modeling a pseudo-realistic case study for software development and identifies the process patterns that are supported by this approach. Finally, this thesis is concluded by presenting the contributions of this research project and discussing the future perspectives.

Part I

Chapter 2 discusses some main principals of software engineering that need to be considered for the evaluation of the context. It presents a discussion on the current standing of software process modeling approaches relative to these notions and their shortcomings. Apart from evaluating the context, this chapter focuses on the targets that are kept in mind while working on this thesis.

Chapter 3 presents most of the common process modeling approaches both from academia and industry. It compares their strengths and weakness keeping in view the characteristic discussed in the previous chapter. It also explains some international standards pertaining to process modeling. A critical analysis of all these approaches backs the motivation of this thesis.

Part II

Chapter 4 illustrates the principle constructs of the process metamodels in each phase of development. It focuses on the refinement approach which enriches these metamodels as the process lifecycle phase advances.

Chapter 5 describes the implementation choices made for this approach. It reasons about different perspectives of the approach regarding the control flow of the framework. It also presents the architecture of the prototype that accompanies this thesis to model the real life processes.

Part III

Chapter 6 presents a pseudo-realistic process for software development by an organization. It describes the modeling methodologies for this process and discusses the strengths and weakness of our approach based on this case study.

Chapter 7 evaluates our process modeling approach against the process patterns to categorize our approach in the context. It presents the results regarding the support of control-flow patterns, resource patterns and data patterns. These results are presented against other approaches for comparison.

Part IV

Chapter 8 presents the contributions of this research work, using the solution criteria defined in the first chapter. This chapter is concluded with the limitations of the current work and the possible future prospects.

Part I

State of the Art

Chapter 2

Software Process Modeling Context

Contents

Abstract *- This chapter discusses the context of software process modeling. Initially, the concept of process is discussed to explain its associated structure and behavior. Then the notion of process modeling and process-centric environments are discussed to explain the different phases of process development lifecycle. Later in this chapter, different aspects of process development are discussed, which are of prime importance for software process modeling. Process reuse, process architecture and process execution are the notions that are highlighted. These notions play an important role in understanding the context of our proposal.*

2.1 Process Modeling

The term 'engineering' in software engineering focuses on the *systematic* and *organized* procedures to carry out the activities for software development domain. As opposed to ad-hoc methods, the target of a procedure in *engineering* is not only to achieve goals, but to accomplish it by following precise and well-ordered tasks. The greater goal of following such methodology is to ensure quality customer value. These well ordered tasks need to be defined before their actual execution. Process models are used to define these tasks and the order in which these tasks needs to be performed in a process. Various languages have been developed to support the modeling of these processes in different contexts. We will not be focusing on the specifics of each process modeling approach in this chapter, instead we would discuss the main concepts of the domain.

2.1.1 What is a Process?

'*Process*' is a general term that has been used in many fields like business process management (BPM), workflow management (WfM), business process improvement (BPI), etc. These fields focus on process as the core controlling methodologies, where the implementations of the business domains are developed around it. They normally term it as '*business process*' and it is defined by Davenport as, *"A structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. Taking a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers"* [Davenport 93]. This definition of process focuses on a general structure and motivation of a business process. A '*software process*' in our view is also a business process that is targeted towards the development of software systems. Specifically, software process is defined as, " *A set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables*" [Lonchamp 93]. So the term business process can be used for software processes in this thesis for two reasons. First, it is a more general term that can explain the concepts. Second, to define software process, the software industry uses BPM technologies, where business processes represent the software processes.

Longchamps's definition can be viewed as an extension to the Davenport's definition of process, where he focuses on a clear process boundary, well-defined inputs and outputs and a structure of action, that transforms the inputs to outputs. Numerous definitions of process can be found in the research domain, but they all focus on related groups of activities, common goals, and the use of people, information and resources [Lindsay 03, Curtis 92]. Level of granularity in the definition of process may vary, but the key concepts are fundamental for the completeness of a process. Processes are defined in detail because adhering to them may be critical for a project's success, specially for the large scale projects [Lehman 91].

Initially, the software engineering community had put a lot of stress on the linear structure of a process, which does not fit well with the software development practices [Lindsay 03]. There has been an argument that workflow view of processes with definable inputs and outputs of discrete tasks, having dependencies on one another in a clear succession is limiting [Keen 97]. So a more flexible definition of a process is *"any work that meets the following four criteria: it is recurrent; it affects some aspect of organizational capabilities; it can be accomplished in different ways that make a difference to the contribution it generates in terms of cost, value, service, or quality;*

and it involves coordination" [Keen 97]. This definition does not explain the structure of a process, neither does it constrain the ordering of activities, it rather focuses on the significant characteristics of a process.

2.1.2 Process Modeling Languages and Notations

Software process programming started to evolve as soon as the software community started to give software processes the same importance as that of software programs [Osterweil 87]. As software enterprise realized that they need to develop their unique processes for each software project, they needed a well defined approach to describe their processes. These processes were later on reused for multiple projects and tailored according to the specific needs of the projects. These process models were required to capture all the details of the product and the organization for developing that product. To respond to this need, Osterweil suggested a notion of '*process program*', that would take into account the process elements needed to describe the work routines of a software enterprise relating to a specific project [Osterweil 87]. These *process programs* gradually evolved into full fledged languages for formal specification of processes, called Process Modeling Languages (PMLs). Curtis et al. [Curtis 92] presented four distinct views to describe these elements modeled by the process programs/models: 1)*Functional View*, that covers the functional dependencies between the processes. These functional dependencies can be input and output dependency, where the output of one process is an input to the other. 2)*Dynamic View*, that covers the control sequencing of the process elements. The control flow and the sequence of processes describe the overall behavior. 3)*Informational View*, that provides the description of work products used or produced by the process. 4)*Organizational view*, that includes the description of the performer of processes and the organizational hierarchy regarding the responsibilities.

The problem with PMLs is the level of detail and formal specification that makes it quite difficult to use in the industry. For this reason, PMLs are mainly used by academia to formally prove various assumptions and characteristics of process modeling. However the research carried out on PMLs gives a formal foundation for high level process modeling notations. The term 'high level' is to demonstrate that other languages use a higher level of abstraction, thus hiding the fine details from the end user. These high level languages can be divided in two categories. First, the Business process modeling languages, that provide the possibility to graphically draw the process flows [OMG 11]. These process flows are used for discussions between stakeholders and for keeping the documentations. Originally they were not meant to be executed, but now with growing influence of IT in business, languages have been presented to execute them [OASIS 07]. The second category is the workflow models, which also allowed to graphically draw the process flows, but were intended to be executed through a workflow management system. Workflow notations are developed for enactment, so they need a well-defined execution semantics. For the development of information systems, the target of the system analysis phase is to understand the process in which the intended system would be deployed. In some recent endeavors, process models are used to describe these processes, which are embedded in the information systems and control their execution [Weigold 12].

2.1.3 Process-Centered Software Engineering Environments

Process Modeling Languages became one of the key research areas of software engineering research and since then new dimensions on process modeling approaches are being explored. The development of Process-Centered Software Engineering Environments (PSEE) are based around the concepts of process modeling. PSEEs are the information systems that provide the notations and mechanisms for the development of process models. These systems also foster the possibility to maintain and enact a process model. PSEE offers support for process management in one or more phases of process lifecycle ranging from requirements specification, assessment and problem elicitation, (re)design, implementation to monitoring and data collection [Ambriola 97]. The PSEE is designed to guide/enforce the user in the development process. The role of PSEEs in guiding a user is classified into four levels from least active to most active as: 1) *Passive role*, that operates on user requests 2) *Active guidance*, where PSEE guides the user 3) *Enforcement*, where user is forced to act as per the direction of PSEE 4) *Automation*, where system does not require user intervention [Dowson 94].

A PSEE offers a PML to support the definition of process models, which are then analyzed and enacted by the environment [Türetken 07]. The analysis of these process models is based on different properties like consistency, redundancy and circularity. The enactment of the process model is handled by the environment according to the degree of guidance provided by the PSEE, where it can demand the user to execute some processes or perform them itself by invoking the related application and IT tools. The focus of PSEE remains on the analysis and enactment of the processes, so they rely on formal languages (PMLs) that are very close to software programs [Taylor 88, Belkhatir 91, Bandinelli 94, Sutton Jr. 97]. Some recent research endeavors targeted the use of process models in PSEE by exploiting MDE [Montoni 06, Maciel 13]. A classification of PMLs based on the support that they provide for a specific phase of process lifecycle and the level of abstraction is provided by Ambriola et al. [Ambriola 97] as:

— *Process Specification Languages (PSL)*, that are used for the requirement specification and assessment of processes.

— *Process Design Languages (PDL)*, that support the design phase of the process development.

— *Process Implementation Languages (PIL)*, that are used for the implementation and monitoring of the processes.

2.2 Process Reuse

Reuse is not a new concept, by any definition. It has been previously used to repeat mathematical models and algorithms across problems to ensure correct calculations [Prieto-Díaz 93]. The origins of software reuse trace back to computer programming languages, where the development effort of software was reduced by the use of libraries containing functions and other reusable units. With the evolution of software engineering, software reuse is no more restricted to code only. Today a wide variety of software development artifacts are reused which range from requirements to design patterns. With increasing competition in information technology, the focus has been diverted to fastest time to market in launching the software, releasing new versions and providing updates. In order to compete with the market, as the business models of the software enterprises changed over time, the methodologies offered by software engineering also evolved. Software development lifecycles evolved from the traditional waterfall model to spiral models, then to iterative models and now agile models. Besides the use of these evolving lifecycle models, software enterprises have been focusing to reduce the engineering effort for developing the software systems.

The efforts and achievements of the past can be reused in software engineering like any other engineering domain, provided the outcomes offer the possibility to do so. Different types of artifacts created during the software development process (*e.g.* requirements, designs, code, tests, documentation, *etc.*) can be reused if they were originally designed for it. Building software from the existing software that has already been developed rather than building it from scratch is termed as software reuse in the domain of software engineering. Software reuse is considered to be one of the foundation principles of software engineering which targets at reducing costs and and minimizing time to market [Estublier 05]. Apart from time and cost, software reuse also aims at improving productivity and quality of the software development process by making it easier to manage the risks, schedule and cost of the project. Designing a software artifact for reuse may increase the cost of development in some cases, but offers a greater pay back in the longer run.

The overall concept behind software reuse is fairly simple. It emphasizes on the techniques and principles that allow the software developers to develop a new system by composing the already developed components that can either be accessed from repositories or purchased. The techniques to reuse an artifact for software development depends upon the type of the artifact *i.e.* reusing a software component is different from reusing a design concept. In terms of abstraction, reuse is classified in two kinds: black box or white box. In black box reuse, the internal implementation of the artifact is unknown to the developer who wants to reuse the artifact. Whereas in white box reuse, the implementation details of the artifact are available to the developer. In terms of planning, reuse is again divided into two types: opportunistic and systematic. An opportunistic reuse of the system artifact is unplanned, where the original artifact was not designed to be reused. On the contrary, for a systematic reuse of the artifact, it needs to be designed for reuse deliberately.

Reuse is not specific to the software artifacts only; other knowledge gained from the experiences and the processes themselves can also be reused. Reuse is defined as *"the use of systems artifacts and processes in the development of solutions to similar problems"* [Whittle 96]. Another more precise definition states, "*In the design of systems, repeated use or application in different places of the design of parts, manufacturing tools and processes, analysis, and particularly knowledge gained from experience; using the same object in different systems or at different times in the same system* [Committee 02]. The motivations for process reuse are the same as software reuse. IEEE Standard 1517 [Society 10] outlines the following benefits of systematic process reuse

— Increase software productivity

— Shorten software development and maintenance time

— Reduce duplication of effort

— Move personnel, tools, and methods more easily among projects

— Reduce software development and maintenance costs

— Produce higher quality software products

— Increase software and system dependability

— Improve software interoperability and reliability

— Provide a competitive advantage to an organization that practices reuse

Some process modeling approaches favor process variability with the intention of promoting process reusability [Estublier 05, Hollenbach 95, Armbrust 09]. One of these approaches presents three types of methodologies for defining and organizing process variations for reuse: enumeration, parametrization and abstraction/inheritance [Hollenbach 95]. Process enumeration is to define multiple processes; one at a time. The process engineer who wants to reuse such a process has to choose manually the process that suits his problem. Process parametrization is used when enough details are known about different process variants and the process engineer can select the desired variant by naming it as a parameter to a given process. For example to estimate the size of a software, different process can be used amongst the established processes: Cocomo, Revic or delphi methods. A process using the estimation process can call it using any of the parameters. Process abstraction focuses of developing generic processes that compose the common features of a set of specific process implementations.

2.2.1 Design by Contract

Design by contract (DbC) is a software construction approach that is based on contracts between the clients (callers) and suppliers (routines) [Meyer 92a]. These contracts rely on mutual obligations and benefits, which are explicitly specified. This approach uses the concepts of Abstract Data Types (ADT) and contracts to bind together the interacting components. ADT ensures the encapsulation of the data and methods behind well-defined interfaces. Contracts binds the client to demand "valid requests" only and the supplier to respond with "valid response" only. This approach exploits assertions to explicitly specify the interfaces for the components through pre-conditions, post-conditions and invariants. This design style was developed in the context of a programming language, Eiffel [Meyer 92b], to create reliable object oriented software. Assertions allow the programmers to make explicit assumptions about the software elements based on their interface specifications. These assertions provides the basic rules which govern the specification of contracts between the interacting components.

The core idea of design by contract is to hold responsible the software elements that are developed to carry out that task. This means that between two interacting software components, it is the responsibility of the client to request the desired service according to the specified contract. Same way, the supplier gives the guarantee (and takes responsibility) to provide the desired output, if the proper conditions are met. These conditions are formally specified in the contract. In case the client or the supplier violates the contract, an exception is raised. The interfaces of these software elements are specified before their actual implementation, and eventually their implementation is guided by the specified interfaces. This guarantees reliable interactions with the implemented components.

Recent research endeavors in process modeling approaches have considered the implementation of processes using design by contract. But their focus is to achieve different targets. A recent approach has targeted to improve interoperability amongst cross organizational business process modeling using design by contract approach [Khalfallah 13]. Another approach focused on the reliability of interactions between multiple parties for a distributed process by presenting a formal definition of interfaces using design by contract [Bocchi 10]. Apart from interoperability, reliability, decomposition and encapsulation, DbC provides two important benefits which support reuse. The first benefit is that the interfaces of a component can be extended in a systematic manner without affecting the existing components. Second, new implementations of the ADTs can be transparently added to the system. In process modeling, a process can benefit from both these advantages to foster process reuse. It can exploit these interfaces to support process variants, that are implementations of the abstract process definitions (process ADTs). And it can also guarantee the reliable interactions between the process elements. In order to support reusability for process models, the process definitions should explicitly specify their interfaces. A reusable process definition should specify its entrance criteria, inputs, outputs and exit criteria through these interfaces [Hollenbach 95].

2.2.2 Interaction between the contracts

Design by Contract ensures reliability of interaction amongst interacting components by explicitly defining the interfaces and the contracts that bind them. The benefit of constraining all the interactions through specified contracts depends on the level of contractual implementation [Beugnard 99]. Beugnard et al. have classified

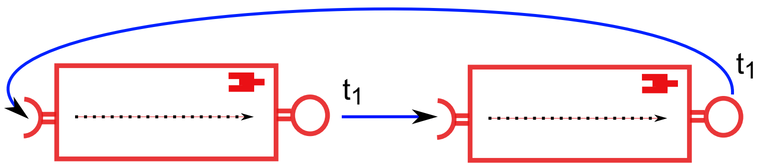


Figure 2.1 – Callback in asynchronous activity components

contracts in four levels: 1) Basic contracts, that focus on structure and makes the system work; 2) Behavioral contracts, that improves the level of confidence in sequential context by exploiting pre-conditions, post-condition and invariants from Design by Contract; 3) Synchronization contracts, that focus on synchronizing the requests in distributed environments; 4) Quality-of-service contracts, that ensure a negotiable level of quality of service.

Software processes are concurrent by nature and may be distributed. In such a situation, only following the Design by Contract is not sufficient to guarantee reliable interactions between the activities. And for making the process components reusable a precise interface specification is of high value. Reusing a process component in an environment where processes are executed simultaneously can cause a deadlock. Szyperski identifies this problem as a call back problem where two components are waiting for a service from each other at the same time [Szyperski 97]. Figure 2.1 explains this problem in the context of a process, where both the activity components are not sequential and are requesting for the service at the same time t_1 and thus result in a cyclic dependency. A technique to solve such an issue is to exploit the notion of *global assertions* to specify constraints on the overall interaction scenario [Bocchi 10]. Szyperski solves this problem by demanding *re-entrance* conditions for such components, but this solves only a special case [Szyperski 97].

This problem is just one of the problems that can be faced in concurrent context with asynchronous components. In classic layered hierarchical system, the callback face much more problems with hierarchical *up-calls* [Giese 00]. For interactions amongst components in arbitrary structured systems, synchronization is of crucial importance [Giese 00]. Use of synchronized contracts (Level-3) can help solve such issues [Beugnard 99]. A precise definition of contracts which specifies the behavior and allows to detect the synchronization issues is of vital importance to guarantee the interaction based on contracts in a concurrent context. Reuse of a component is not effective if reliable interaction through the interfaces is not guaranteed. Hence, to foster reusability in process components, their interfaces should be specified with fine details.

2.3 Process Architecture

Various researchers have focused on the similarities between software programs and process models [Osterweil 87, Vanderfeesten 08]. The reason for this is the structural similarity in them. Both of them can be partitioned into smaller modules, where

- ADL
  - Architecture Modeling Features
    - **Components**
      - **Interface**
      - Types
      - Semantics
      - Constraints
      - Evolution
      - Non-functional properties
    - **Connectors**
      - Interface
      - Types
      - Semantics
      - Constraints
      - Evolution
      - Non-functional properties
    - **Architectural Configurations**
      - Understandability
      - Compositionality
      - Heterogeneity
      - Constraints
      - Refinement and traceability
      - Scalability
      - Evolution
      - Dynamism
      - Non-functional properties
  - Tool Support
    - Active Specification
    - Multiple Views
    - Analysis
    - Refinement
    - Code Generation
    - Dynamism

Figure 2.2 – Taxonomy of ADLs. Courtesy: [Medvidovic 00]

each module takes some input, processes it and gives out the output. Software program uses the concepts of functions or methods, whereas processes have activities. Similarly modules in software programs are made up of operations, whereas activities are made up of tasks. Furthermore there are a lot of resemblances in terms of interactions, execution order, updating the values of data objects, *etc.*

The similarities between software programs and process models have led to process architectures, that are inspired from software architecture. Software architecture languages focus on the description of the building blocks (components) of the system, their assembly and the interactions between these components. Software architecture in itself is a very diverse domain with diverse terminologies. Figure 1 presents a taxonomy used for comparative analysis of different software architecture languages for holistic picture of the domain [Medvidovic 00]. The essential modeling elements of software architecture are components with their interfaces, connectors and the architectural configurations.

The notion of component in software engineering is very general but there is a common agreement that it is a unit of computation or data store [Medvidovic 00]. Even in the domain of component based development, a design time component is different from the implementation time component and then it has entirely different semantics during instantiation. In software architecture, a component is a module that has a defined boundary and specified interface, which is a set of interaction points between it and its external or internal context. Connectors are the units of coordination between the components. They model the interactions between the components and the rules that govern these interactions. Finally, the architectural configurations are the structural 'connection' arrangements of components and connectors to form a system. These configurations are responsible to make sure that appropriate components are connected, their interfaces match properly, connectors are coordinating the interactions properly and the their combined semantics result in the expected behavior of the system [Medvidovic 00].

2.3.1 Architectures for process modeling

Inspirations from the software architecture lead to the notions of components (process elements) and configurations in process architecture. A component in process architecture may represent an activity or a primitive task. Interfaces of the activities are generally described as inputs or outputs of an activity. These inputs and outputs of activities are the work products. Configurations for the process in general can describe a variety of notions from sequential flow of activities to contractual interactions between them. Processes are made up of other processes which in turn might be a collection of processes. Thus process are inherently hierarchical in nature.

CMMI specification [Team 10] defines process architecture as, "*A 'standard process' is composed of other processes (i.e., subprocesses) or process elements. A 'process element' is the fundamental (i.e., atomic) unit of process definition that describes activities and tasks to consistently perform work. The process architecture provides rules for connecting the process elements of a standard process*". And then further in the specification they specify different kinds of rules that are used for defining relationships amongst the process elements as:

— Order of the process elements

— Interfaces among process elements

— Interfaces with external processes

— Interdependencies among process elements

It should be noted here that the first rule, that emphasizes the ordering of process elements in CMMI specification, does not restrict to use either explicit or implicit ordering. How much focus should be given to the sequential information of process elements? This question brings us to a choice between imperative or declarative process modeling language.

2.3.2 Declarative vs Imperative Process Modeling

An important architectural choice is to place the process modeling languages in the spectrum of imperative versus declarative languages. Empirical studies classify the existing process modeling approaches in this spectrum based on the extent to which a process modeling language relies on *sequential* or *circumstantial* information [Fahland 09, Pichler 12]. Another distinction between the both paradigms for process modeling is described as, "*Procedural [Imperative] models take an 'inside-to-outside' approach: all execution alternatives are explicitly specified in the model and new alternatives must be explicitly added to the model. Declarative models take an 'outside-to-inside' approach: constraints implicitly specify execution alternatives as all alternatives that satisfy the constraints and adding new constraints usually means discarding some execution alternatives*" [Pesić 08].

The possibility to follow a *continuous* forward (or backward) trajectory from a place (state) or transiton in a Petri Net to see the process behavior places it in imperative process modeling languages [Fahland 09]. Same way different flow based languages are imperative, whereas the event based languages, where the focus remains on 'what' rather than 'how' are declarative. The choice of imperative vs declarative in a process modeling language does not hinder its usage, however they do effect the understandability of the process. Tasks that contain sequential information are better represented using an imperative style whereas declarative style is more understandable for tasks that contain circumstantial information [Pichler 12].

2.3.3 Service oriented architectures in process

The problems faced by the software enterprises are heterogeneity in terms of systems, application and technologies. Integration of these technologies is of high value, when complex processes are to be dealt with. Efficient business decisions in competitive environments are based on instant information access and data integrity. An integrated information system helps in reducing the time for information access and provides data integrity across the complete system, hence supports decision making [Papazoglou 07]. In order to integrate these heterogeneous sub-systems, the development of the overall architecture has to be based on interfaces for communication. A service-oriented architecture (SOA) that is based on interfaces allows developers to overcome many distributed computing challenges like application integration, transaction management, management of security policies, management of legacy systems and all this by using different platforms and protocols.

Conventional software architectures tend to structure the organization of the system in its sub-systems (components) and the relationships between them. On the contrary, SOA designs the software system around services, which are provided either to the end user applications or other services distributed in the network. These services are propagated through the network using published and discoverable interfaces [Huhns 05]. So the service oriented architecture is basically based on the web services architecture model, shown in figure 2.3.

Services in SOA are packaged software resources that are well-defined and self-contained modules that provide standard business functionality. They promote loose coupling by being independent of the state and context of other services. They are described using web services standard e.g. Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), Universal Description Discovery and Inte-

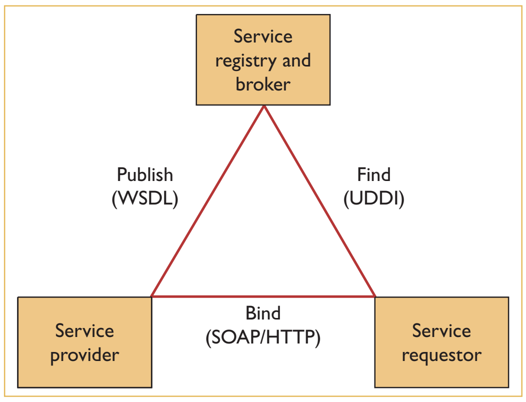


Figure 2.3 – Web Services architecture model [Huhns 05]

gration registry (UDDI), etc. Web service definitions are based on published interface through which they communicate with each other and request the execution of their operations. Multiple web services operate together to collectively support a common business process. "*The purpose of this architecture is to address the requirements of loosely coupled, standards-based, and protocol-independent distributed computing, mapping enterprise information systems (EIS) appropriately to the overall business process flow*" [Papazoglou 07]. Besides the architecture of a SOA, its lifecyle is also very different from the traditional systems. In other software systems once the system is modeled, CASE tools are used for code generation, whereas for SOA this code generation is replaced by service discovery, selection and engagement [Huhns 05]. A workflow of web service development and execution in service oriented architecture is illustrated in figure 2.4.

Software processes in large enterprises are not developed to cover a whole spectrum of activities, which may or may not be performed within the enterprise. Software sub-contracting is a very common activity in the current business models. The nature of software development allows its implementation in a distributed environment. Cross-enterprise interactions or even the distributed development of software within an enterprise requires the capability to enact software processes at geographically separated locations. Hepp et al. position web services and business processes together as the future of software application structures [Hepp 05]. The business process management and web services communities have already presented a number of languages and standards that describe business processes from the perspective of web services orchestration. The most prominent language in this context is WS-BPEL [OASIS 07], which serves for the enactment of business processes in distributed environments.

2.4 Process Execution

There is an increasing trend for adopting process driven methodologies in software development enterprises. Ideally, all the different tasks performed in a software enter-

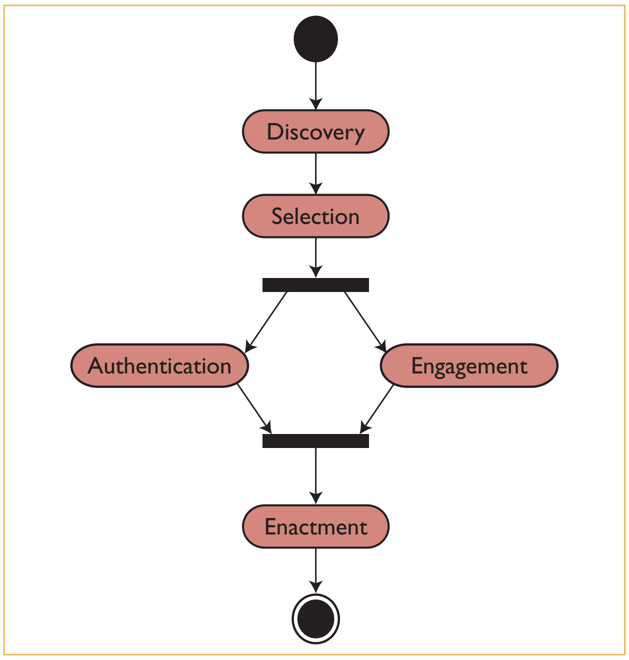


Figure 2.4 – Web Service development and execution workflow [Huhns 05]

prise are explicitly defined so as to promote standardization and improve their control, flexibility and effectiveness. In order to deliver quality customer value, these processes are often supported or at times fully implemented by software systems [Rossi 07]. A process execution typically involves various applications, services and humans. Specialized process management systems are developed to integrate and control these processes to achieve the desired business goal. These process management systems rely on the inherent behavior of the process modeling languages. For industrial use, typically two types of process management systems are in use: Workflow Management Systems and Business Process Management.

2.4.1 Workflow Management Systems

Workflow management coalition defines a workflow as, "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" [Coalition 99]. A workflow is a process model that is used to describe process definitions by modeling the contained activities, procedural rules and associated control data to manage its execution. Each process instance in a workflow has its own specific set of data associated to that individual process instance. In order to execute these processes, a workflow management system is required. Workflow management coalition defines a workflow management system (WfMS) as, "*A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition,* *interact with workflow participants and, where required, invoke the use of IT tools and applications*" [Coalition 99]. WfMS offers a number of functions that are used to define and graphically monitor workflows. Basically three types of functions are defined by WfMS [Hollingsworth 95]: 1) built-time functions, that are used to define the workflow through processes and activities; 2) run-time control functions, that are used to manage the workflows during execution; 3) run-time interaction functions, that are used as an interface between workflow activities and human users or IT tools. Based on the wide adoption by the community, different vendors like Microsoft [1] and IBM [2] and open source projects like Eclipse Stardust [3][Eclipse 13] are offering Workflow Management Systems.

Implementation of the workflow coordination primitives is realized by a workflow engine. Workflow engine interprets the (graphical) workflow representations using a computations form, known as workflow description language. From an architectural perspective for deployment, there are two types of settings that can be used for the workflow management and coordination: the centralized architecture and the decentralized architecture. For a centralized workflow, a single workflow engine serves as a dedicated coordinator that is responsible for routing the messages between the business partners. This routing is based on the patterns of data flow between the activities, as described by the workflow. On the contrary, for a decentralized workflow management, each business partner implements a local workflow engine that manages the workflow for that partner locally. The overall workflow management and control tasks are distributed between these business partners. The interactions between the business partners are based on the data flow patterns of the workflow, and is managed between the local workflow engines.

2.4.2 Business Process Management

Service oriented architecture provides a standard, loosely coupled and interoperable structure for mapping the enterprise information systems through the use of application services, as explained in section 2.3.3. Web services offer a suitable technical foundation for the business process community to develop their process models in a fashion that they can be accessed in a distributed environment, within and across multiple enterprises. This also allows the process models to access operations of information systems that are geographically separated. Business processes are implemented as web services. Processes are designed with reusability in mind, having start and end points. Reusability in this context is the ability to execute a process repeatedly. Web service composition is exploited in a way that business processes use other web services to carry out a task [Tan 09].

Business process management is a holistic approach that covers all the phases related to a process in the business process management lifecycle. For business process execution phase, Web Services Business Process Execution Language (WS-BPEL,

1. `http://msdn.microsoft.com/en-us/vstudio/jj684582.aspx`
2. `http://www-03.ibm.com/systems/power/software/i/workflow/`
3. `http://www.eclipse.org/stardust/`

BPEL for short) has played the most significant role [OASIS 07]. BPEL is a language for defining and executing business processes. It is based on web services and it exploits the web services composition, orchestration, and coordination for realizing SOA. It offers the possibility to standardize the business processes for interoperability. Apart from this, it helps in business process optimizations and offers the capability to select the appropriate process at runtime. As BPEL has become a de facto standard for executing processes as services, multiple vendors like Microsoft [1], IBM [2] and Oracle [3] have developed their own BPEL engines to support process driven software development. We also find open source suits that provide BPEL engines for process execution like jBPM [4].

2.4.3 Process-driven Applications

Processes are developed and executed to automate the software development methodologies. In order to do so, the process engines are connected with the application tools and services. This way, the complete Information system can be integrated and controlled around the process engine. In such situations, there are two methods to empower the process engine to control/help in controlling the rest of the development environment. One of the ways to automate software development is to embed the process engine within the software application as a component. This component is then bound to other components of the system that provide the actual functional code, as shown in figure 2.5-A. This way process definitions represent the main control logic of the application and process engine is responsible for triggering other components. Applications developed with this architecture are called process-driven applications[Weigold 10].

Figure 2.5-B depicts the architecture followed by the process management systems (BPMS/WfMS), where the business process engine is implemented as a standalone software system that interacts with other software applications, services and humans to achieve the business goal. This architecture is not very domain specific and allows a generic process engine that can be used with different types of applications in multiple domains.

2.4.4 Process Execution Concerns

When we discuss software process modeling approaches in the context of industrial development, the most commonly used approaches are BPMN[OMG 11], workflows[Coalition 99] and SPEM[OMG 08]. Process models developed through workflows can directly be interpreted by the Workflow management systems, using the workflow definition language. On the contrary, if BPMN and SPEM are used, they do not allow a direct interpretation for the executions engines. An intermediate

1. `http://www.microsoft.com/en-us/biztalk/default.aspx`
2. `http://www-01.ibm.com/software/integration/wps/`
3. `http://www.oracle.com/technetwork/middleware/bpel/overview/index.html`
4. `http://www.jboss.org/jbpm/`

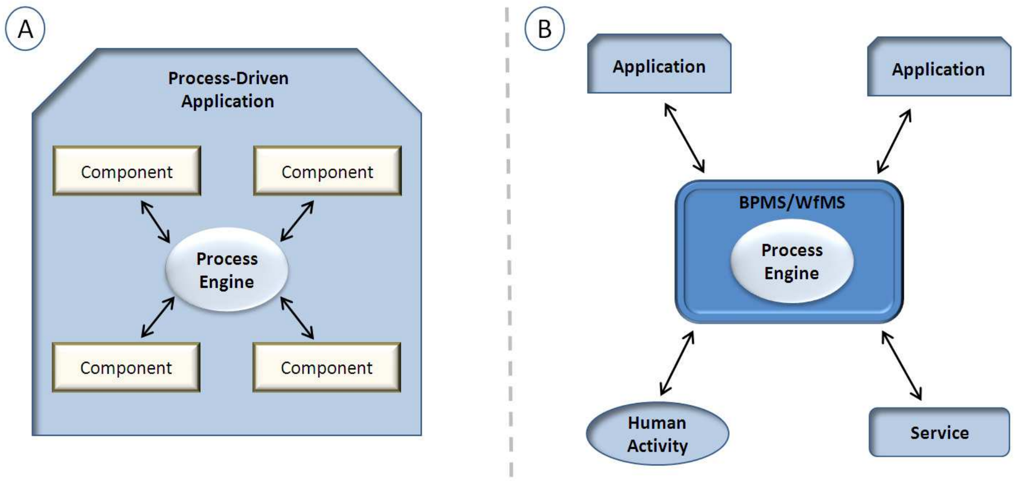


Figure 2.5 – Process-driven applications vs. BPMS/WfMS [Weigold 10]

transformation is required to map the process elements of these languages to BPEL. There is no standard interchange language to map BPMN constructs to BPEL, however academic research proposes a transformation between them [Ouyang 06]. BPMN is a collection of multiple diagrams associated to process modeling, but lacks a standard interchange between them. This makes the mapping of BPMN to BPEL even more complex. A recent standardization effort from OMG targets this problem, and they have formed a BPMN Model Interchange Working Group [1] to develop a standard interchange language. Same way, SPEM standard leaves out on mapping it to any process execution language, rather suggest a mapping to any project management suite[OMG 08]. However, another effort from academia, xSPEM extends the original SPEM metamodel and enriches it with the operational semantics and adds the possibility to map it to BPEL, using WSDL [Bendraou 07].

Another important concern related to the execution of software processes is to manage the deviations. Real life processes in software development projects may not follow the exact plan. In such cases, the Process-centered Software Engineering Environment (PSEE) observes inconsistency between the software process model and its actual execution, which is termed as deviation. Multiple approaches have been presented to detect and handle these deviations [Kabbaj 07, Almeida da Silva 11]. On the part of process modeling approaches, they offer mechanisms for process tailoring [Hurtado Alegría 11] and run time process adaptability [Kabbaj 07].

2.5 Shortcomings of the process methodologies

Let us clearly distinguish between two different concepts: process models and process modeling methodologies. The methodologies are used to develop process models.

1. `http://www.omgwiki.org/bpmn-miwg/doku.php`

When we do a comparison between different approaches we need to compare the methodologies themselves and what those methodologies produce. So two questions can be posed in this regard: How effective and systematic are the process modeling approaches? and, How simple and complete is a process model? When we argue that BPMN lacks a consistent methodology to offer transformations of its implementation process models to the current enactment methodologies like BPEL, it is a shortcoming of the methods that it offers. This reduces the scope of a BPMN model for process automation, but it offers what it has to offer within its scope. On the other hand, BPMN process models has their own limitations. For example it is not possible to associate multiple roles with an activity in a BPMN model i.e. an activity can not reside in two swim-lanes.

Understandability of process models is of prime importance for software enterprises [Indulska 09]. As process designers are different from process performers, they may develop the processes much before they are actually used. Software practitioners suggest to have as few number of elements to model the processes as possible, for their understandability [Mendling 10]. One of the reasons for the lack of adoptions of Process Modeling Languages (Process Programs) in industry was the level of detail and formalism that they required. This is why the software practitioners moved toward the graphical representations of the process models. Abundance of process modeling concepts in a graphical notation can clutter the model and can effect the understandability to a much higher degree than that of PMLs. Excess of modeling notations should not be confused with the completeness of the process models, which can be achieved even through a minimum number of concepts.

Model should be independent of the person who is modeling it [Rosemann 06b]. Specific expertise of software practitioners influence the way they visualize a problem and then further model it. We believe it to be one of the reasons that business process modeling focus on the flow based ordering of activities. A process modeler should develop a model in a way that it is complete enough to model that process, otherwise he/she can end up with developing a view of the process model rather than a complete model. It should be noted here that a process modeler is not obliged to develop a complete process model, because the focus on relevance is of much more importance than completeness. However, depending of the situation a complete process model can be relevant and thus a process modeling approach should always offer methodologies that can ensure completeness. Another important issue related to completeness is the ability of the process enactment environment to enact the process models even if they are not complete. This may be important for some situations where early simulation is required or the process details are expected late and can be added to the process model during enactment [Sutton Jr. 97].

As the process model should offer the capability of adaptations and evolution, so should the methodology [Rosemann 06b]. A software process modeling methodology targets at defining the processes of software development in an enterprise. But for an enterprise, their may be other points of interest that can be linked with process models. For example if an enterprise wants to model risks associated with certain processes along with the process models, they should have the capability to customize the approach and the associated tools. A software enterprise may be interested in modeling other issues like cost and knowledge management as well. Extensibility of the approach is possible if the architecture of the approach implemented in the tool support is accessible and offers extension points. Even for the open source tools, the architecture of the process management system is hardly available.

The most important motivation for a software development enterprise to model its processes is process improvement [Indulska 09]. In fact, following a process modeling approach that does not allow (or does not provide means to support) process evolution can be quite harmful for a software enterprise in the sense that it can demotivate for innovation [Andersen 01]. Process evolutions mechanisms are also of prime importance to handle runtime process deviations [Kabbaj 07]. Process evolution (adaptation) mechanisms provided by the standard process modeling approaches both in business process management and workflow management are not sufficient enough to deal with the needs of industry [Rosemann 06b]. However some academic proposals have targeted this deficiency to enrich the software process models [Kabbaj 07, Almeida da Silva 11].

When we talk about the reuse of process models, it means that the process model should allow the possibility to be used in some other iteration (having same or different context) within the same project or in another project. This means that maintenance of this process model is also an important issue, related to its (re)use. Current modeling methodologies offer the possibility to define associated roles with the processes, but do not offer the roles associated to the process element of the model. For example a role of test engineer is assigned to perform the activities defined in a process model for software testing. This process model does not offer any role of a process designer who could be the possible owner of this process, and is responsible for its design and implementation. Another associated role is of the process engineer that is responsible for enacting and maintaining the process model. Lack of governance (process ownership) is one of the reasons that limit the applicability of process models in real life situations [Rosemann 06a].

Chapter 3

Process Modeling Frameworks

Contents

Abstract - *This chapter presents the state of the art in software process modeling. Process modeling approaches are categorized in two groups, where flow based languages are the process languages where the flow of work products is the prime focus and event based languages are the languages that keep the notion of events as the main concept for process component interactions. These process modeling languages are described individually and are evaluated against the solution criteria defined in section 1.3. Industrial standards relating to process modeling are also discussed.*

3.1 Introduction

The term 'process modeling', as we use nowadays, emerged from the concepts of office automation, that led to the definition of 'workflows'. Business process management became popular as late as in 90's, which established the ground for process modeling. Different approaches have been proposed since then to model processes either before or after their execution. Software process management approaches target to streamline the information systems development and facilitate the automation of process flow. One of the common perspectives to see process modeling is as "*step-by-step rules specific to the resolution of some business problem*" [Havey 09]. Approaches following this perspective focus on the flow of activities in a process model. Some of these techniques are outlined in section 3.2. Instead of focusing on model readability and modeling human activities, Havey argues that the quality of a process modeling language is measured by the level of its contribution to the information system design and model execution potential. For these reasons we have chosen some of the prominent existing process modeling approaches that are directly executable or can be mapped to some other execution language.

We do not discuss the software lifecycle models like Rapid Application Development (RAD) [Martin 91], the Spiral Model [Boehm 86] and XP [Beck 99] in this state of the art. We have instead restricted our focus to the Process modeling approaches that focus on the structure, semantics and lifecycle of individual processes rather than on software lifecycle. Approaches for evaluating software lifecycle models (design process models) present a taxonomy to classify them according to their salient characteristics [Céret 13]. This taxonomy can also be used to evaluate the software process models described in this chapter. However, we have evaluated them against the solution criteria identified at the beginning of this thesis.

Out of the various process modeling approaches, we chose some of the approaches that are prominent and are adopted by the industry. We also present some academic approaches that extend the functionality of some prominent approaches. The reason for not covering other approaches in this evaluation might be either of the two; their approach somewhat resembles to one of the approaches we are already discussing or they are focusing on entirely different axes. ADEPT2 process management system and Kinesthetics Extreme present executable process models that allow runtime deviations [Göser 07, Valetto 01]. They focus on the reactivity of the process models and offer exception handling capabilities. They use some of the exception patterns to jump backwards or forwards in the control flow in order to deal with exceptions. In order to discuss the reactive aspects of process modeling, we discuss EPCs, YAWL and Little-JIL. However, the proposed approach does not offer exception handling, so we did not discuss these approaches in detail. One of the examples of the approaches that focus on a different axis is presented by Cortes-Cornax et al., where they focus on the development of a mapping between process models and goal models, in order to use goal-oriented requirements engineering techniques to analyze goal satisfactions [Cortes-Cornax 12].

3.2 Flow based Approaches

We have categorized the process modeling languages that focus on the flow of data from one activity to another in order to model the complete process, as flow based approaches. These languages may present the notion of events, but it is not the central notion for interaction between the activities.

3.2.1 Software Process Engineering Metamodel (SPEM 2.0)

SPEM 2.0 is presented by OMG with the vision to provide a process modeling approach that can deal with projects that are specific to software development. SPEM specification defines itself as , "*a process engineering meta-model as well as*

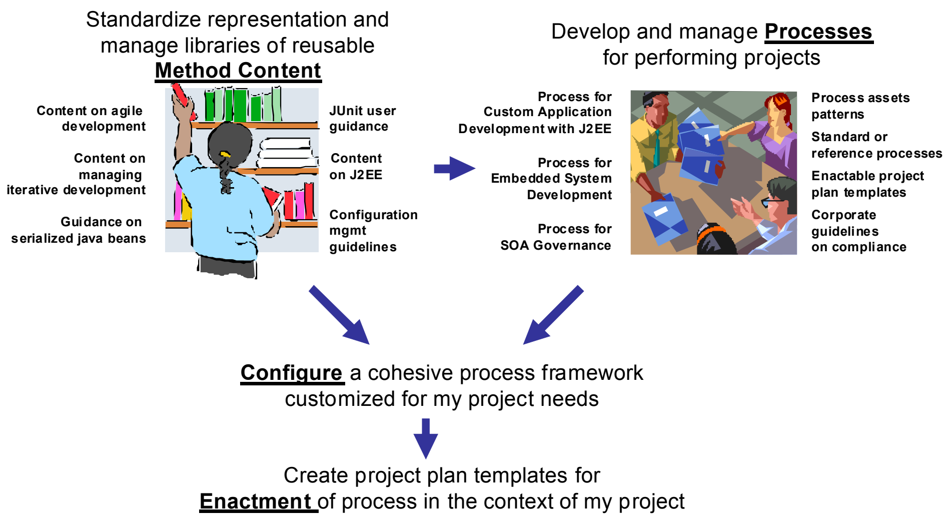

Standardize representation and manage libraries of reusable Method Content
Content on agile development
Content on managing iterative development
Guidance on serialized java beans
JUnit user guidance
Content on J2EE
Configuration mgmt guidelines
Develop and manage Processes for performing projects
Process for Custom Application Development with J2EE
Process for Embedded System Development
Process for SOA Governance
Process assets patterns
Standard or reference processes
Enactable project plan templates
Corporate guidelines on compliance
Configure a cohesive process framework customized for my project needs
Create project plan templates for Enactment of process in the context of my project

Figure 3.1 – SPEM 2.0's conceptual usage framework [OMG 08]

conceptual framework, which can provide the necessary concepts for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes. An implementation of this meta-model would be targeted at process engineers, project leads, project and program managers who are responsible for maintaining and implementing processes for their development organizations or individual projects"[OMG 08]. SPEM specification presents itself in two flavors: MOF 2.0 compliant metamodel that reuses parts of UML2.0 and as a UML profile.

The SPEM 2.0 (hereafter called SPEM) replaces its earlier version SPEM1.1, by providing some important notions. The most important of them is to separate process contents and materials from their usage in a specific project. This philosophy is explained in the conceptual framework of SPEM that presents the overall methodology in four steps (illustrated in figure 3.1). This framework divides a process model in two parts (step 1 & 2). First, the method contents are defined *i.e.* activities, artifacts, roles, tools *etc.* And then in a second step, instances (or references) of these method contents are used to assemble a process definition. Last two steps are for configuration and enactment of the process model.

SPEM metamodel is presented using seven packages. Each package merges in the package below it till the final `Core` package (figure 3.2). `Core` package presents the abstract classes that define work (*e.g. WorkDefinition*) and the classes that allow use to define user-defined qualifications of SPEM classes. `Process Structure` package defines the core hierarchy of activities that form process models. Each activity has its own lists of references to *Roles* and input/output *WorkProducts* (as *Role Uses*, *WorkProduct Uses*). These *Roles* and *WorkProducts* themselves are defined in the `Method Content` package, along with other content elements that are used to assemble process models. For assembling a process model, structures from `Process Structure` package are linked with the `Method Content` package, using the notions

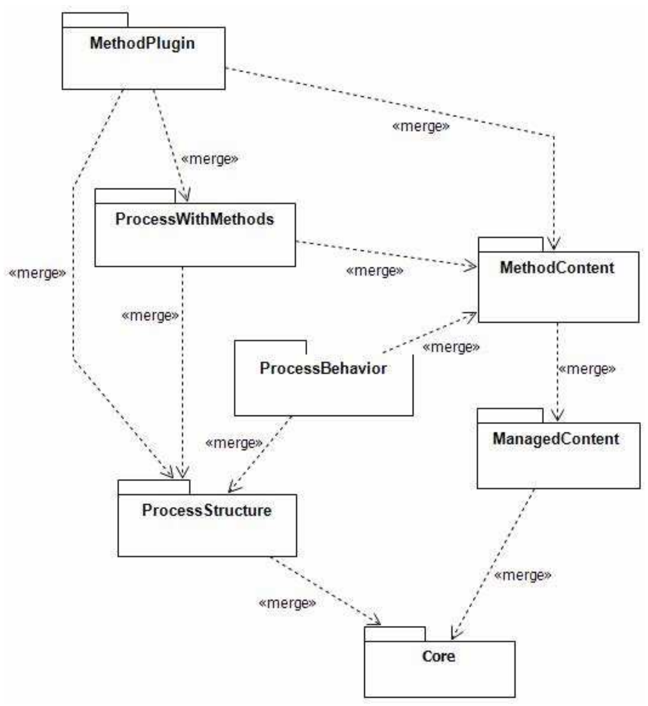


Figure 3.2 – Structure of the SPEM 2.0 Meta-Model [OMG 08]

defined in `Process with Methods` package. `Process Behavior` package allows the possibility of extending the structures presented in `Process Structure` package with (externally-defined) behavioral models. Textual documentation of the process models is made possible through the concepts defined in `Managed Content` package. Finally, `Method Plugin` package offers the mechanisms for managing configurable repositories of method contents and processes.

As discussed earlier, SPEM aims at separating the method contents from the process activities. The method contents in this regard are the *Role Definition*, *Task Definition*, *WorkProduct Definiton*, *etc.* The method contents are defined to build a development knowledge base. To use them in an activity of a particular process, they are referenced through the concepts of *Role Use*, *Task Use*, *WorkProduct Use*, *etc.* A *Role Use* is defined as "*a special Breakdown Element that either represents a performer of an Activity or a participant of the Activity*" and a *Role Definition* as "*a Method Content Element that defines a set of related skills, competencies, and responsibilities*" [OMG 08]. So in order to develop a process model, initially the *Role Definition* has to be defined and is placed in the knowledge base. Later on, to define the activity in a particular process model, a Role Use is used as a reference to the *Role Definition.* The notion of *x-Definition* and *x-Use* should not be confused with any type-instance relationship or any conformance relationship. *x-Definition* is a definition of a modeling element stored in the knowledge based, whereas *x-Use* is a reference/pointer to it.

Guidance is a *Model Element* associated with the major *Model Elements*, which contains additional descriptions for practitioners such as techniques, guidelines, procedures, standards, templates of work products and so on. A Discipline is a Process Package organized from the perspective of one of the software engineering disciplines: configuration management, analysis and design, test, and so forth.

SPEM does not offer any reactive control and the ordering of activity is based on the dataflow between the activities. The flow of data between two activities creates a dependency in them. This dependency is defined through the notion of *WorkSequenceKind*, which can be either of start-start, start-finish, finish-start or finish-finish. A notion of *Process Component* is also presented by the SPEM specification, which ensures encapsulation of a single activity. Each process component defines its input/output *WorkProduct Ports* as its interfaces. These process components are once again bound together through the *WorkSequenceKind*. The notion of component in SPEM is primarily used for exploiting encapsulation. It serves when the process model does not define the implementation of an activity, hence uses a (blank) blackbox process component, which can be replaced later on during execution by another process component that defines its implementation. A SPEM process model defined by process components does not follow the design by contract approach [Meyer 92a] as it does not focus on the interfaces for the purpose of interactions. SPEM specification proposes two methods for enacting a process model, either with project planning systems or with workflow engines, but it does not provide a standard mapping to either of the two.

SPEM framework has been adopted by many tool vendors for software process modeling for three main reasons. First, by separating the method content and process structure it allows the software enterprises to develop their knowledge base of intellectual capital. Second, it allows to create catalogs of pre-defined processes. Third, it does not constrain the tool vendors with predefined constraints on the behavior of process model, which makes it easy for them to support SPEM along with other process modeling approaches within the same tool. Several tools implement the SPEM framework. Eclipse Process Framework (EPF) Composer[1] is a SPEM compliant process modeling tool. StarUML[2] is another open source UML modeling tool that supports the development of SPEM models in its new version. IRIS Process Author[3] is a visual process management system supporting SPEM, that offers software process development, improvement and automation.

Solution criteria based evaluation

The conceptual framework of SPEM gives an overall idea of the approach, where abstraction between definition and usage of process elements are at its core. It describes the necessary concepts for defining a process model, that is based on the flow of *WorkProducts* between the activities. *Breakdown element* is an abstract general-

1. `http://www.eclipse.org/epf/`
2. `http://staruml.sourceforge.net/en/`
3. `http://www.osellus.com/IRIS-PA`

ization for any type of process element that is a part of *breakdown structure*. This notion of breakdown structure is used to describe hierarchies in process elements.

Completeness: SPEM specification allows the definition of the basic constructs of *what* (WorkProduct Definition), *who* (Role Definition) and *how* (Task Definition) in a very interesting manner. However the constructs related to *why* (goals, intentions, *etc.*) and *when* (scheduling, planning, *etc.*) are not addressed in the specification. The notion of *pre-condition* serves well to describe the completion criteria for individual tasks or activities, but does not guide the overall intent of the system. No specific behavior is defined for SPEM metamodel (with the intent to keep it flexible), which might be the reason to avoid focus on these aspects. For example, no constructs are defined for managing the control of activities in terms of loops or conditions, *etc.* and it is left for the tool implementer to choose any control flow for the coordination of activities. Same way, it completely misses out on any mechanism for communications (choreography) between the agents.

Team Development: The notion of *Team Profile* is used as a breakdown structure for Role Uses and Composite Roles, which are important to structure teams. Each Role Definition is described as a collection of *qualifications* to define its competencies and skills. However, a Role Use can be either responsible for a task/activity or its participant (responsible of a sub-activity). There is no classification of responsibilities associated to a task. For example, a task can be associated to three roles with different responsibilities, at the same time: performer, approver, consultant. Apart from the definition of a well structured team, a process modeling approach should also focus on the distributed environments, specially in software development scenarios where outsourcing is a common practice. Well defined interfaces for activities supports the possibility to integrate/distribute sub-processes to multiple process owners (organizations). These aspects are not addressed in SPEM specifications.

Reusability: The main focus of SPEM lies on the separation of method content from the process structure. This allows to define the method contents for once and then (re)use them in assembling different specific process models. The notions of 'element definition' and 'element use' are interesting, but it could offer more if the semantics of the relation between the both would be of conformance. An approach with a conformance relationship between process elements can induce variability in addition to reusability [Golra 12a]. Another approach for process reusability described by the specification is to reuse process patterns either through "*sophisticated copy and modify operation*", the details of which are very implementation specific. This method of process reuse is more opportunistic than systematic.

The reuse of process elements is also targeted with the notion of process components in SPEM. Process components are taken as black box entities that encapsulate the *Activity Use / Task Use* in way that inputs and outputs are possible only through the specified interfaces. *WorkProduct Port* as an interface to a process component only provides the dataflow capabilities to the process component. It adds modularity to the process model, but it is not guaranteed as the behavior defined by the implementer can break its interfacing for tool invocations, interactions, control flow, etc. So the reusability of process components in SPEM implementations relies on the tool implementations of SPEM. This means that reusability is not part of the framework in this regard but more of an implementers choice.

Abstraction: The SPEM metamodel itself is built on the principle of abstraction where seven different packages, each describing a different logical unit, are merged in the *core* package. Tool vendors can choose amongst three of the compliance points given by it to implement all or part of the given packages. SPEM specification allows the extension of process models through the notion of *Activity Use Kind* property. This property allows an *Activity Use* to use the *Activity Definition* as it is, or to extend it. SPEM does not exploit the notion of abstraction to a greater depth. For example, one is not able to develop multi-level process models like in Situational Method Engineering [Gonzalez-Perez 07] or evolution-based process models[Jaccheri 93].

Modularity: Modularity of process models developed using SPEM specification is discussed by presenting the notion of process components. As discussed earlier, this concept is far from the notion of software components, however it serves well to modularize the process models. SPEM metamodel does not elaborate on the interactions between process elements or the *Role Uses* associated to these process elements, neither does it explain the interactions with system (*e.g.* tool invocations). It remains the responsibility of the third party implementer to keep the modularity intact, which is not guaranteed by any constraint provided by the specification. Hierarchical modularity is well defined in SPEM where all key process elements are generalizations of *Breakdown Element*. *Roles Uses* and *WorkProduct Uses* are also breakdown elements. Processes, activities and tasks are *WorkBreakdown Elements*, which allows to create hierarchies of these elements.

Tailorability: The conceptual framework of SPEM, illustrated in figure 3.1, shows that the third step in developing a process model is to configure it according to the project specification. This configuration allows a controlled process tailoring because the method contents already developed in the knowledge base can be reused in process models. Multiple process element definitions of the same kind can be developed and stored in the knowledge base, which can be replaced with the current ones. However, the specification does not take into account any details related to runtime process adaptations.

Enactability: The specification does not offer any concepts for enactment. It does not even describe the notion of states for activities or the transitions between these states. Even the notion of actor, a person performing a role, is not defined. However it suggests two examples of enacting a process model. First, it suggests that it can *"be systematically mapped to a project plan by instantiating the different Process' breakdown structure views"* [OMG 08]. Second, it suggests to use the enactment machines of different behavior model approaches after mapping the process elements to the specific behavior model elements. No formalisms are provided for mapping SPEM process models to any behavioral model. Thus different approaches have been proposed to extend SPEM metamodel to add execution semantics [Bendraou 07, Koudri 10a, Portela 12]. As a consequence, there is no standard mechanism for process enactment under SPEM specifications. Two SPEM metamodel extensions (xSPEM and MODAL) that add the execution semantics to it through different mechanisms are presented in the following sections.

3.2.1.1 xSPEM

xSPEM metamodel is an extension to SPEM and is presented through four metamodels: Domain Definition, State Definition, Events Definition and a generic Trace Management metamodel [Bendraou 07]. The complete architecture of xSPEM is illustrated in figure 3.3. The Domain Definition metamodel is presented through two packages: 1) *xSPEM_ Core* package, that reuses a minimal set of process elements from the *Core* and *Process Structure* packages of SPEM2.0 metamodel like *Activity, Role Use, WorkProduct Use etc.* 2) *ProjectCharacteristics* package, that introduces properties to process elements relating to activity scheduling and resource allocation [OMG 08]. These properties include a time interval for activity, role occurrences and workload for a role. States Definition metamodel presents *ProcessObservability* package, which enriches the process elements with the notion of state and defines the behavioral semantics for process execution. Events Description metamodel is described by the *EventDescriptions* package, which defines the events that can trigger activity state changes. Finally, the Trace Management metamodel keeps track of the sequence of events during execution.

The behavioral semantics defined for xSPEM is validated through transitional semantics. The target technical space chosen for the transitional semantics validation is timed Petri nets. The semantics of xSPEM is defined as a mapping to Petri nets. The properties of SPEM2.0 are evaluated by translating them into linear temporal logic, LTL properties on a corresponding Petri net process model for validation. Instead of developing a process engine for the xSPEM metamodel, they chose to enact their process model through a mapping into BPEL [OASIS 07]. The inability of BPEL to model human activities is well known [Schall 08]. For these reasons, they also propose to use a mapping towards BPEL4PEOPLE [Kloppmann 05].

Solution criteria based evaluation

xSPEM is presented as an extension to SPEM2.0, where the main structural core is reused. For these reasons, adoption of xSPEM does not offer any improvement in terms of abstraction, modularity and reusability. We focus our evaluation on the criteria targeted for improvement by this approach.

Completeness:

SPEM specification did not take into account the notions relating to the resource allocation and activity scheduling for process models. *xSPEM_ ProjectCharacteristics* package of xSPEM extends the *xSPEM_ Core* package, which in turn extends the *Core* package of SPEM. xSPEM then uses the concepts of Role Use and WorkProduct use as resources which are allocated to the activities. The process activities are enriched with properties that help in scheduling and workload management. A significant

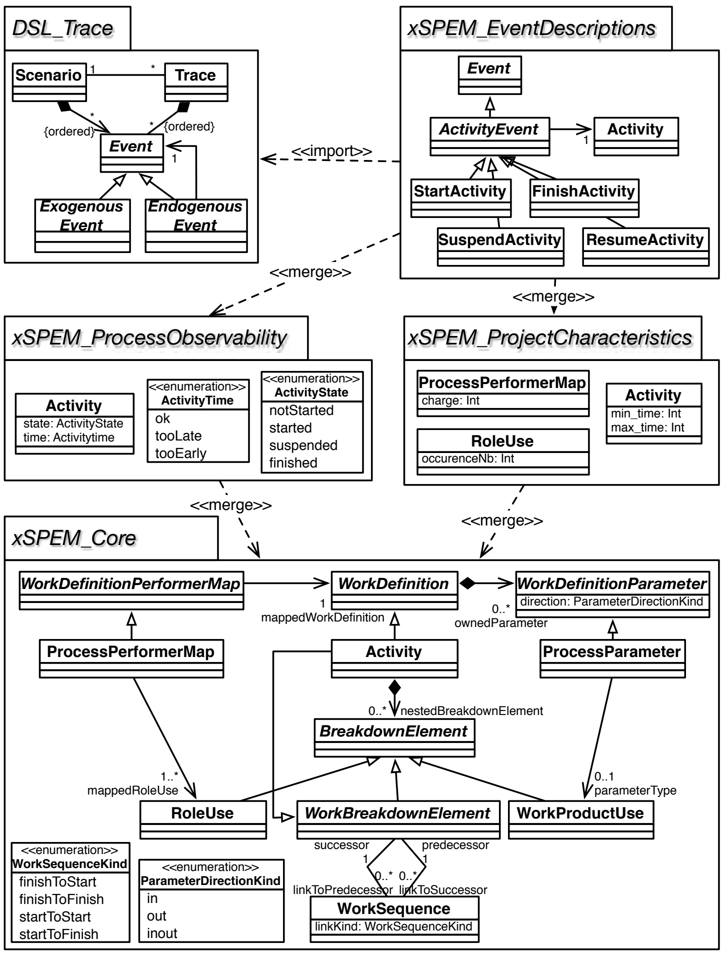


Figure 3.3 – The xSPEM metamodel [Bendraou 07]

improvement over SPEM is to provide the behavioral semantics with the possibility of model checking process models. However the control flow operators for defining logical flows (like join or merge) and loops (iterations) are still missing.

Team Development: One of the benefits of choosing BPEL for process enactment is the possibility to take advantage of the distributed access provided by service oriented architecture. This significantly improves the possibilities for enacting the process in a distributed environment. For a process modeling approach, interactions between the human agents is very important for team development. However, this approach does not take care of this requirement. Addition of properties like number of role occurrences to perform an activity and workload calculation for each role are an improvement step in this direction.

Tailorability: This technique has targeted the execution of SPEM based process models. SPEM itself offers the basic capabilities of process tailoring. However one of the things missing from the SPEM specification was the runtime adaptation of the process models, due to the lack of support for execution. xSPEM does add the execution support, but does not offer any possibility to adapt the processes at runtime.

Enactability: Definition of the behavioral semantics for process execution is the key focus of xSPEM. A mapping towards BPEL is presented for process enactment. The constructs of states and events provide the basis for the execution semantics of the approach. States are defined for the activities in a very rigid four-states automata, without offering any possibility to extend it. The human aspects of the process execution like actors are not very elaborated. Interactions between these roles is also not taken care of.

3.2.1.2 SPEM4MDE

SPEM4MDE is an endeavor to extend SPEM2.0 metamodel for MDE domain [Diaw 11]. SPEM describes the design concepts of a process but does not focus on the MDE concepts for a process. These concepts are added to the language through SPEM4MDE. It also allows the possibility to execute the processes, an important consideration that is not addressed by SPEM. SPEM4MDE attempts to define the execution semantics for the process models using QVT.

It reuses the concepts defined by three different OMG standards: SPEM2.0, UML2.2 and QVT, as shown in figure 3.4. The metamodel is structured using three packages: *MDE Process Structure*, *Model Relationship* and *MDE Process Behavior*. *MDE Process Structure* package use the SPEM structures to define activities and model transformations (as particular activities). The *Model Relationship* package explains the relationships between the models like composition and refinement. The *MDE Process Behavior* package describes the behavior of MDE process elements using UML state-machines. It also describes the execution semantics of transformations using *QVT Base package*. An important concept covered by this language is to define *TransformationDefinition* as an activity. It is described through informal rules, input & output models, and source & target metamodels. The relationship between the constructs of the source and the target metamodels is described through informal rules. As other SPEM activities, roles are specifies through *RoleUse*, which are performed by *ProcessPerformer*.

Solution criteria based evaluation

SPEM4MDE is presented as an extension to SPEM2.0, where the main structural core is reused. For this reason, adoption of SPEM4MDE does not offer any improvement in terms of abstraction, modularity and reusability of the process models. However, the use of MDE concepts allow to use abstraction and reusability for the models that are treated as inputs and outputs of the process. Thus these concepts add help in the execution of the processes in MDE domain, but do not effect the development of process models themselves. We focus our evaluation on the criteria targeted for improvement by this approach.

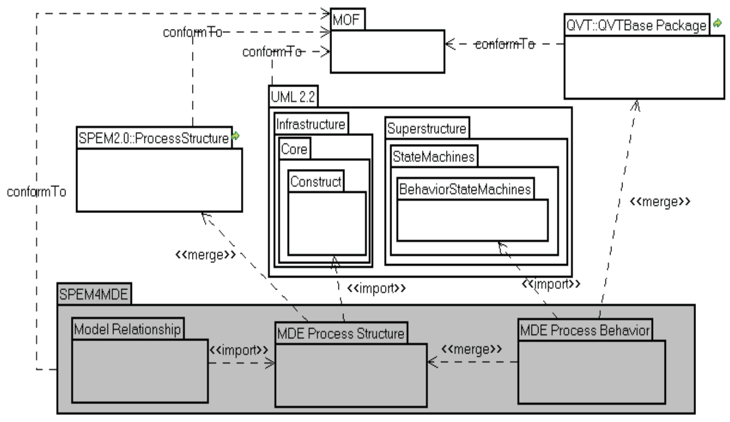


Figure 3.4 – SPEM4MDE packages [Diaw 11]

Completeness:

SPEM specification describes a general concept of activity that is not very specific to the activities used in MDE domain. The core of MDE domain is based on the concept of model transformations. This is taken as one of the most used activities when following the MDE paradigm. Its definition in SPEM supports the input models and the output models but fails to associate it to the deeper concepts like tranformation rules that connect the constructs of the associated metamodels. In this regard, SPEM4MDE completes the SPEM specifications for defining model transformations. Concepts like *InitialActivity* and *FinalActivity* (pseudo-activities) add to the execution semantics of a process where start and end points of a process in execution are defined. However the control flow operators for defining logical flows (like join or merge) and loops (iterations) are still missing.

Team Development: This approach does not improve any of the team development aspects of process modeling of SPEM. However, it defines the model transformation activities in the same manner as SPEM, thus associating Roles, RoleUse and ProcessPerformer to them. A restriction for automatic execution of such activities limits to use a single ProcessPerformer for each transformation definition. Each ProcessPerformer can only be linked to a single RoleUse. Other RoleUses of tranformation activity that could have served for transformation definition testing, acceptance, etc. are not considered. The same research team has worked on another extension of SPEM, that caters the problems of collaborative development, not originally addressed by the SPEM specifications [Kedji 12].

Tailorability: This technique has targeted the execution of SPEM based process models. SPEM itself offers the basic capabilities of process tailoring. However one of the things missing from the SPEM specification was the runtime adaptation of the process models, due to the lack of support for execution. SPEM4MDE does add the execution support, but does not offer any possibility to adapt the processes at runtime.

Enactability: A prototype accompanied with the approach, SPEM4MDE-PSEE serves for the development and enactment of the process models. SPEM4MDE process editor allows to develop the process models. These process models are developed with well specified behavior. This editor may also be used for adapting the process models. SPEM4MDE Process Enactment Engine allows to enact these process models. It can be used to keep track of the states of each process element. This enactment engine is integrated with other eclipse-based tools to allow the execution of model transformations. The outcomes of the model transformations are stored in a project repository.

3.2.1.3 MODAL

The inability of SPEM2.0 to provide sufficient executable semantics has been a motivation for various research teams to add this capability to the original metamodel [Bendraou 07, Portela 12]. These approaches offer a mapping of SPEM metamodel elements towards a standard project management suite or an executable process framework like one offered by BPEL [OASIS 07]. Instead of providing a mapping, Model Oriented Develoment Application Language (MODAL) presents a metamodel extension to SPEM metamodel by enriching it with behavioral semantics and using model transformations to generate executable process models [Koudri 10a]. MODAL introduces/refines some concepts of SPEM so as add some rigor in the process models to reinforce their semantics.

The core metamodel of MODAL is an extension to the *Method Contents* package of SPEM reusing the concepts of *Task Definition*, *Role Definition*, *WorkProduct Definition*, *etc.* [Pillain 11]. It adds the concepts of *Tool Definition* for the specification of tools and their integration with the process models for their possible invocations to support process execution. The *Process Component* in SPEM metamodel has a very vague definition, which does not allow its direct mapping to any component execution platform. MODAL refines the process components by replacing the *WorkProduct ports* with more refined ports that offer services to other components. MODAL allows concurrent execution of process components that are bound together through *Service Bindings*. The notion of lifecycle has been added to the meta-classes related to work products that gives them a *state*, which is defined through a state machine. This also helps in managing models as work products during execution.

A concept of intention is introduced in MODAL to keeps track of the set of methodological objectives set by different stakeholders for some activity [Koudri 10a]. These intentions are linked together through satisfaction links to create intention maps. Intention maps can be refined from coarse-grain to fine-grain through the use of *strategies*. A strategy helps in choosing a particular intention map for the realization of the process in a particular technical space. Constraints of the SPEM model are refined with formalisms that can guard the execution of the process models. These constraints specify their level of severity, based on which they can be relaxed.

Solution criteria based evaluation

MODAL is an approach that extends SPEM for enriching it with mechanisms to generate executable process models. The extensions to SPEM mostly cover the key notions like activities, process components and constraints. Some new concepts are also added like intentions and strategies. Overall these refinements and the new concepts effect different aspects of the language that we are going to discuss. But we would leave the aspects that have not been affected by this extension like team development.

Completeness: MODAL has reused the key process elements from the *Method Content* package of SPEM as *Role Definition* and *Task Definition*. It adds a new notion of *Tool Definition*, which was missing from the original SPEM specification. SPEM specification defined *Tool* in its *Process Structure* package for tool usage, but did not offer the possibility to specify tools in detail. It was also not possible to perform tool invocations, a concept related to process enactment. SPEM does not propose any behavior for its process models and offers a package of proxy behavioral classes, that can be extended to add behavior. MODAL did not follow SPEM specifications in this regard and added the behavior in two parts 1) by adding constraints to guard the execution of process models that are equipped with state machines for activities and 2) by proposing an action language that describes the internal behavior of process components. The notion of lifecycle in MODAL is more inclined towards the state transitions, and the planning and scheduling aspects of process models are not elaborated. Notions of actors and their organization that will be performing the roles while enactment are also not discussed. A notion of connector is defined for process components, but the simple control logic for workflow (like AND, OR, etc) still remain missing. Finally, no mechanism for Role-interactions or communication between actors is offered.

Reusability: The extension to SPEM metamodel has been carried out in a very systematic manner, where the separation of *Method Contents* and *Process Structure* (as describe by SPEM) are considered. New concepts for tool specifications are added as *Tool Definition*, which keep the original idea of reusability offered by SPEM. The most interesting improvement in terms of reusability is the concrete definition of process components. Process components defined by SPEM offered their interfaces but only for work products, which reduced the scope of their reusability. On the other hand, MODAL proposes a concrete concept of interfaces apart from the notion of ports. All interactions to and from the process components are constrained to be through these interfaces, which improves the reusability of process components.

Abstraction: The process models developed under the MODAL approach utilize the concepts of abstraction in two manners. First, the concept of intention associated with every activity and the satisfaction links that connect them together form an intention map. These intention maps are linked to the specific technological spaces through strategies. Thus two levels of abstractions in terms of process plans are used, which are refined through the use of particular strategies. Second, the definition of process models is carried out in three levels. Abstract level process components are defined in *Abstract Modeling Level*. Then the topology of the execution platform of the application is added in *Execution Modeling Level*. Finally, detailed descriptions for a fine grained analysis of the execution platform are added in the *Detailed Modeling Level*.

Modularity: Modularity of SPEM process components has been effectively improved by MODAL. The definition of detailed specification of process interfaces allows to decouple process components. SPEM process components only allow *workproduct* ports, and the rest of the interactions of process component are not defined through the interfaces. The use of *Services* offered through ports ensure proper encapsulation and a better modular approach.

Tailorability: The level of tailorability offered by the SPEM specification is kept intact in this approach. MODAL keeps the separation between *method contents* and *process structure*, thus allowing to update the *x-use element's* reference with some other *x-definition's element*. Support for runtime adaptation of process elements is not considered in this approach. No mechanisms to transfer the state between the runtime replacements is devised.

Enactability: Process model instances in MODAL framework are simulated using COMETA [Koudri 10b], a language defining model of computations for the simulation of hierarchical concurrent component communications [Pillain 11]. This virtual platform allows the definition of execution semantics for the process model. COMETA models are then transformed into executable Java programs. An action language is presented to model the internal behavior of process components through a flow of executable actions. Three main actions are used *Send Action*, *Receive Action* and *Execute*. The tooling support for the process model presents three main components: 1) a MODAL editor to define process models, 2) an instantiation editor to add process behavior and instantiation properties and then transform the model to a COMETA program, and 3) a simulator to simulate COMETA process instances on a Java Virtual Machine. This approach does not elaborate on the instantiation properties and how to manage the issues of resource management, human resource management, tool invocations, *etc.*

3.2.2 Business Process Model and Notation (BPMN)

BPMN is developed by BPMI [1] and chosen as a standardized notation for business process modeling by OMG after the merger of both organizations [OMG 11]. Its development is based on some former modeling approaches like UML, IDEF, ebXML, RosettaNet, LOVeM and EPCs [Recker 06a]. The author of the first specification of BPMN explains two main considerations for the development of BPMN. First, to provide a notation that is easy to use and understand by business users of different level of technical competence, ranging from business analysts to technical developers. Second, to offer an expressiveness to model complex business processes that can be mapped to business execution languages like BPML, which was later replaced by

1. `www.bpmi.org`

BPEL [White 08]. Since then, the vision of BPMN2.0 (hereafter called BPMN) has remained the same as to "*provide a standard visualization mechanism for Business Processes defined in an execution optimized business process language*" [OMG 11]. Keeping in view the simplicity of notation, BPMN specification marks the following aspects as out of the scope.

— Definition of organizational models and resources

— Modeling of functional breakdowns

— Data and information models

— Modeling of strategy

— Business rules models

BPMN was originally developed as a graphical grammar to complement (initially BPML, and then) BPEL standard so that it can bridge the gap between business design and execution. Due to this reason, the constructs defined in BPMN had to cope with business process design and their execution as well, which is normally handled by technically advanced users. To be able to present such a language for users with different level of technical background, the specification divides the BPMN constructs into two sets of graphical elements. The first set targets at providing a very basic notation that can be used to model abstract business processes easily. The second set is an extended set for detailed process modeling that covers complex process scenarios and formal requirements.

The complete BPMN specification defines forty-three distinct grammar constructs along with their attributes. The elements are categorized in five basic groups *i.e. Flow Objects*, *Data*, *Connecting Objects*, *Swimlanes* and *Artifacts*. All other groups except *Connecting Objects* are nodes. These nodes are linked together through *Connecting Objects*. The most basic elements are the flow objects like *events*, *activities* and *gateways*. They offer the basic structural nodes of the process models. An activity is a work performed within a process and is the central notion of a BPMN process, as shown in figure 3.5. Activity types are sub processes, tasks and call activities. Tasks are the primitive activities of BPMN that can not be refined to a more finer level of detail. *Sub processes* are the activities that are further modeled using other tasks, events and flows *etc.* that account to a complete process. Each *Call activity* serves as a reference to a global process and its activation transfers the control to the referenced global process. *Event* is defined as an occurrence that has an impact on the flow and are distinguished as start, intermediate and end events. Gateways are used to converge or diverge the flows.

Data group is represented by four constructs: *Data Objects*, *Data Inputs*, *Data Outputs* and *Data Stores*. Data inputs/outputs are the data objects that serve as work products. They can be created, manipulated and used during the execution of the process. Each data item has an associated state. *Connecting Objects* are represented by *sequence flows*, *message flows*, *associations* and *data associations*. *Sequence flows* depict the order of flow elements in a process whereas *message flows* show the flow of messages between the participants. *Associations* link flow elements to the artifacts

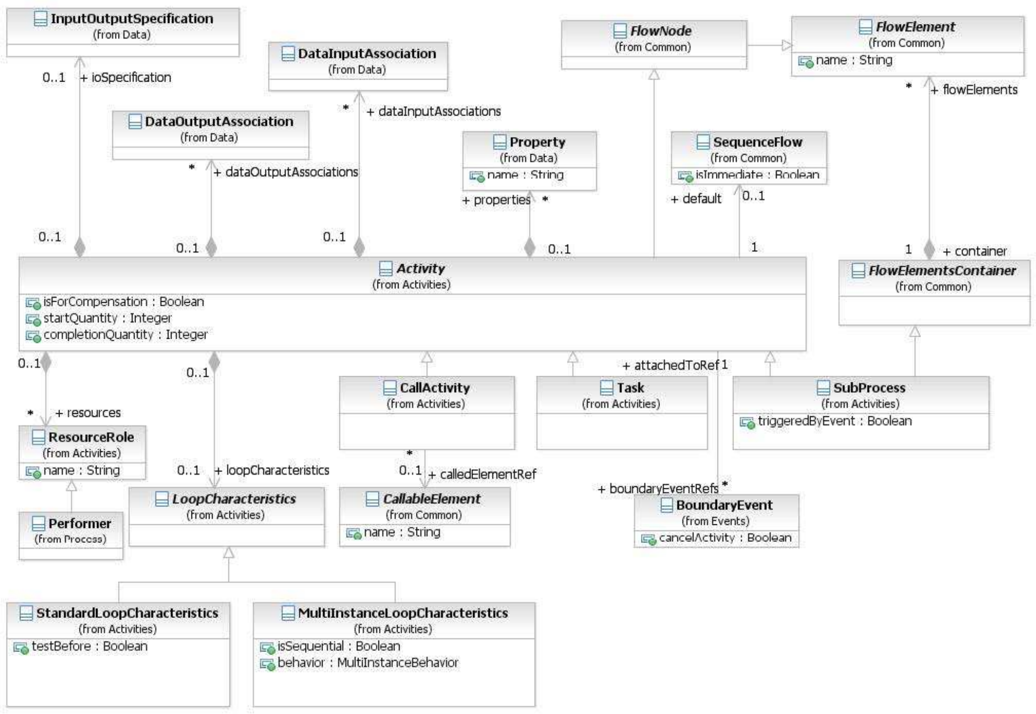

InputOutputSpecification (from Data)
DataInputAssociation (from Data)
DataOutputAssociation (from Data)
FlowNode (from Common)
FlowElement (from Common)
name : String
Property (from Data)
name : String
SequenceFlow (from Common)
isImmediate : Boolean
Activity (from Activities)
isForCompensation : Boolean
startQuantity : Integer
completionQuantity : Integer
FlowElementsContainer (from Common)
CallActivity (from Activities)
Task (from Activities)
SubProcess (from Activities)
triggeredByEvent : Boolean
ResourceRole (from Activities)
name : String
Performer (from Process)
LoopCharacteristics (from Activities)
CallableElement (from Common)
name : String
BoundaryEvent (from Events)
cancelActivity : Boolean
StandardLoopCharacteristics (from Activities)
testBefore : Boolean
MultiInstanceLoopCharacteristics (from Activities)
isSequential : Boolean
behavior : MultiInstanceBehavior
+ ioSpecification
+ dataInputAssociations
+ dataOutputAssociations
+ properties
+ default
+ flowElements
+ container
+ attachedToRef
+ resources
+ loopCharacteristics
+ calledElementRef
+ boundaryEventRefs

Figure 3.5 – Activity Class Diagram of BPMN [OMG 11]

and *data association* to the data objects. *Swimlanes* represent *pools* and *lanes*, where pool is a graphical representation of a participant or organization and lanes are used to sub-partition them. Finally, *Artifacts* are used to provide additional information regarding the process. BPMN specification provides two artifact types: *groups* that are used to categorize graphical objects and *text annotations* that are used to provide additional textual information for graphical objects.

Because of the wide scope of business processes, an effective business process modeling approach has to cover different point of views related to the same business process. BPMN describes these different points of view using 3 main diagrams: *Process Orchestrations*, *Choreographies* and *Collaborations*. Processes (orchestrations in SOA context) can be categorized in two groups, private and public. A private process definition depicts the business processes that are internal to an organization, which may or may not be executable. Executable processes can be executed through a formal definition of semantics, whereas the non executable processes are developed for documentation purposes. Private business processes are developed in a single pool of swimlane, where they are not authorized to cross its boundaries. However, public processes depict the interaction of processes to the context (other processes or participants). These interactions are modeled through the *Collaboration* diagram. A collaboration diagram uses two or more *pools* representing multiple public processes, where message exchanges are depicted through *Message Flows*. A *choreography* is also a depiction of interaction between two participants, where no pools are described. A choreography describes the interaction behavior amongst the participants in a procedural way. BPMN 2.0 added another view of choreography, *conversation*, which is an informal description of a particular usage of collaboration diagram.

Solution criteria based evaluation

BPMN specification is defined to bridge the gap between the design and execution of process modeling. It defines the necessary concepts for modeling a business process. Even though it is more inclined towards the business aspects of a process, it provides sufficient details for the technical aspects of a software process model. We evaluate this approach with respect to the needs of the software development domain.

Completeness: Even though BPMN is a business process modeling notation that does not specifically target software development projects, it presents a lot of concepts that are sufficient for the basic need of a software process modeler for defining the structural aspects of the process. However, some specific modeling concepts of software development need to be modeled as a work around e.g. the notion of pre/post conditions of an activity need to be worked around through the used of conditional sequence flows. A post-condition of an activity ensures the proper completion of an activity rather than the invocation of the next activity. The human resource management aspect of BPMN is also lacking the necessary details to manage the actors that play the roles. *PartnerRoles* and *PartnerEntities* describe the current role played by the participant or organization, but do not provide the possibility to assign some specific responsibility to a role. The specification defines the execution semantics of BPMN and has also added a mapping towards WS-BPEL in its latest version.

Team Development: Collaboration diagrams and conversation diagrams focus on the message flow between the participants of a process. They help solve many issues relating to team development of process models. However the notion of swimlanes for describing the roles and teams is not flexible. For example, it is not possible to associate multiple roles with an activity unless they are all members of the same team. It is not advisable to make separate teams for every activity that use multiple roles. Because of the lack of standard interfaces between processes, we argue that it is hard to integrate the sub-processes that are not built for the current process. For example the public process of two separate enterprises have no common contract to follow, which makes it difficult to adapt them for collaboration.

Reusability: Keeping opportunistic reuse aside, BPMN does not focus on any design principles that can offer a systematic reuse *i.e.* design for reuse capability. An activity in BPMN model is associated with sequence flows, message flows, events, artifacts, data objects without any formal interface specification. This results in a tight coupling of activities, which reduces the possibilities of reuse.

Abstraction: The abstraction mechanisms used in BPMN allow to hide the details of a process by collapsing it. BPMN specification allows to collapse the sub processes and activities in a process model, which can be expanded for its concrete implementation. The notion of *call activity* provides a reference activity which transfers the control to a global process upon invocation. This is another mechanism to abstract the fine details of a process. However, abstractions in terms of conformance relationships between process elements are not presented.

Modularity: BPMN specification allows to modularize the processes in terms of hierarchy. The use of *sub processes*, *activities* and tasks ensures this hierarchy. However we feel that it is very constraining for a modeler to develop one block with exactly one entry and one exit. No activity or process in BPMN process orchestration is allowed to have multiple entry 'ports' and multiple 'exit' ports. On one hand, it makes it easier to specify the formal semantics to avoid deadlocks and lack of synchronization. But it restrains the process modeler to build flexible process models without using extensive hierarchies. Lack of standard interfaces for activities does not allow to modularize a system in a loosely coupled architecture.

Tailorability: We did not find any specific notion that supports process tailoring in BPMN specification. BPMN does not provide its own process engine to execute the process models directly. They need to be transformed to BPEL models, so the dynamic adaptations of the processes is out of the scope of BPMN. However, due to the semantic differences of BPMN and BPEL, traceability of BPEL elements back to BPMN is not effective [Ouyang 06]. This makes it quite hard to tailor a process once it is transformed to BPEL.

Enactability: One of the extensions provided by BPMN 2.0 over BPMN 1.2 was to define the execution semantics of its process elements. This execution semantics helps in realizing the mapping towards BPEL. The specification provides a mapping of BPMN model to BPEL model. BPMN specification does not provide a mapping of complete BPMN diagrams to BPEL, rather it provides a mapping between BPMN orchestrations and individual WS-BPEL processes, where each BPMN orchestration concerns only one pool. BPMN presents a much richer semantics than BPEL so the mapping between them is not always trivial. "*Not all BPMN orchestration Processes can be mapped to WS-BPEL in a straight-forward way. That is because BPMN allows the modeler to draw almost arbitrary graphs to model control flow, whereas in WS-BPEL, there are certain restrictions such as control-flow being either block-structured or not containing cycles*" [OMG 11].

3.2.3 Business Process Execution Language (WS-BPEL)

WS-BPEL is an XML-based language for specifying business processes and the model governing their operation in the web service environment. BPEL is a collective term used for both BPEL4WS Version 1.1 and WS-BPEL Version 2.0 [OASIS 07]. It uses several XML specifications like WSDL1.1, XML Schema 1.0, XPath 1.0 and XSLT 1.0. The data model used by BPEL is provided by WSDL and XML Schema, whereas the data manipulation is handled by XPath and XLST. External resources and partners are represented through WSDL services. Each partner (process) exposes a WSDL interface with at least one port type for being eligible to be included in the overall composition. The relationship between a partner service and a WS-BPEL business process is realized through a mandatory Partner Link. Each partner link has

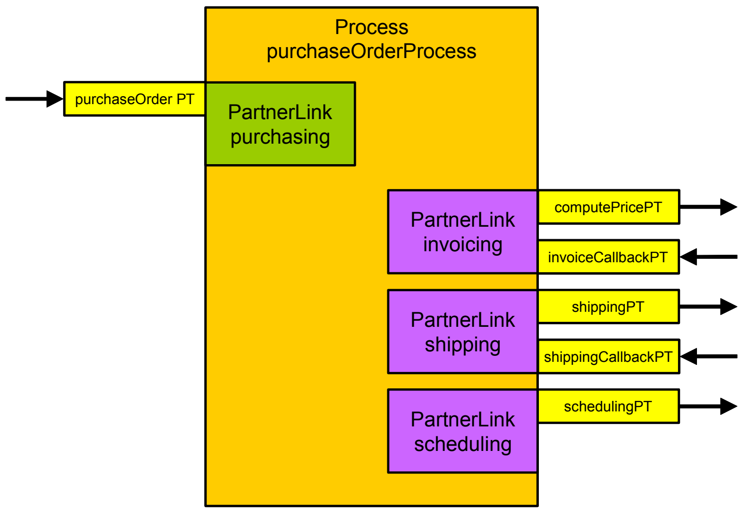


Figure 3.6 – BPEL process structure [OASIS 07]

up to two roles and declares which port type each role requires for the interaction to be carried out successfully. The structure of a BPEL process with partner links and port types is shown in figure 3.6. For a complete understanding of the example presented in this figure, readers are suggested to read the BPEL specification [OASIS 07].

BPEL is defined around the idea of building business processes from the invocations of existing web services and their interactions with external partners. Business processes model the actual details and behavior of a participant in an interaction. This internal behavior of the process is kept hidden in interactions. Process descriptions (abstract processes) specify the business protocols that only describe the mutually visible message exchange behavior of the involved partners [Havey 09]. A BPEL business process is defined using two files: 1) A BPEL file, that presents the 'stateful' definition of the process through its activities, partner links, variables and event handlers, and 2) WSDL documents that specify the 'stateless' web service interfaces (required and provided services) for the business process defined in the BPEL file. A BPEL document is structured in XML and is influenced by the concepts of web services [Ko 09]. The core elements of the BPEL document are:

— roles of process participants

— port types for interactions between participants

— orchestration, that defines the flow of the process

— correlation information, that defines the manner in which the messages are routed to the correct composition instances.

Process logic is described in the process definition through activities (XML elements), which are of two kinds: basic and structured. Basic activities represent the actual 'functional components' of the process and include <invoke>, the <receive>/<reply> pair, <assign> and <wait>. These constructs are used to describe the elemental steps of process behavior through web service interactions. On the other hand, structured activities describe the control structures like other conventional programming languages. They include constructs like <if>, <while>, <repeatUntil>, <pick>, and <foreach>. Parallel execution is supported through the <flow> element, where the order of execution can be controlled using <link> elements. In addition to the activities, BPEL specifies handlers for events and faults. Every handler has an associated event, scope and a corresponding activity to handle the event [Ko 09]. The state of a process is represented through BPEL variables, which are of three types: WSDL message type, XML-Schema type and XML-Schema element.

Solution criteria based evaluation

BPEL is a language that targets the execution of business processes through the use of web services. Being an XML-based language it is already very verbose. So as to be human understandable, it needs to describe the executable processes with minimal concepts. We evaluate BPEL as a process execution framework, not as a process modeling framework.

Completeness: BPEL specification offers the basic constructs to develop a process model through the use of basic and structured activities. The idea of using structured activities gives it a block-like nature, which is very close to the conventional programming languages. The use of if structures, switch structures and looping constructs makes it far away from the high level modeling languages [van der Aalst 05b]. BPEL is acyclic in nature, whereas the real life processes may contain cycles of activities like a review cycle of some activity until its acceptance [van der Aalst 05b]. The same author also advocates that BPEL's abstract process only depicts the perspective of one side of the collaboration. One of the issues with BPEL is the lack of expression due to the lack of constructs. For example, if a BPMN process model is transformed into BPEL, it undergoes a considerable semantic loss [Recker 06b].

Team Development: The inability of BPEL process models to deal with the human aspects of process execution is a known issue [Kloppmann 05]. BPEL processes are well supported to deal with automatic activities presented as web services. These services can be invoked by a process and a composition of multiple services makes up the process. On the other hand, human processes can not be invoked in the same fashion. Extensions to BPEL like BPEL4PEOPLE provide support to deal with human processes in an effective manner [Kloppmann 05]. Besides this, as BPEL focuses on single perspective of the collaboration [van der Aalst 05b], it is hard to depict a two way choreography between the participants.

Abstraction: *Abstract process* in BPEL is a process description meant to describe the message exchange behavior of the participants. It should not be confused with an abstraction of the process model. A process model is defined as a *business process*, which describes the internal behavior of the process. These two concepts deal with the 'private' and 'public' behavior of a process. However, web services provide abstraction inherently through the use of service compositions. A web service may provide its functionality by invoking (abstracting) many other web services. Thus a

BPEL process is made up of process hierarchies, where one process invokes another process to contain it.

Modularity: Web services are designed in a fashion that they expose their interfaces for interactions between heterogeneous systems. BPEL uses the same methodology and exposes its WSDL interfaces for communicating with other processes. This ensures the decoupling of business processes and thus the overall process description offers a sufficient level of granularity for distributed environments. However, for the development of a single web service that does not compose other web services, the concept of modularity is not exploited well. For example, there is no concept of BPEL fragments that can be invoked from within the same or from different BPEL processes [Ma 09].

Reusability: The essence of using a web service is to 'build of reuse'. Different BPEL processes can reuse (by composition) another process. This reuse of services is valid for the coarse level of web services (processes). However, apart from this level, when we talk about the development of a single web service that does not compose other web services, we explained earlier that it does not exploit modularity. As a consequence, the reuse of BPEL processes is also restricted to the web service level and code fragments (*scope* or BPEL fragments) within a single BPEL process are not reusable.

Tailorability: Web services provide a very modularized architecture to BPEL. This architecture is exploited well by BPEL to offer tailorability. A process contains many other processes/activities to offer its services. Any of the sub-processes of a BPEL process can be replaced by any other process that conforms to the required interface specification. This helps in adapting the process even at runtime, by replacing its building blocks. However for a primitive service that does not use any other service, no specific mechanisms for tailoring its interfaces is provided.

Enactability: BPEL uses service oriented architecture to enact business processes. Each business process is taken as a web service that might invoke other business processes (services). The choreography between the processes is handled through the defined interfaces. BPEL is considered to be a suitable process enactment for the automatic processes, however it has its limitations for dealing with human processes. As human processes can not be invoked in the same manner as other services, they need to have a separate mechanism. Some extensions to BPEL like BPEL4PEOPLE have been proposed to deal with such issues [Kloppmann 05]. The acyclic nature of BPEL makes it hard to model the processes that rely on multiple iterations. Real life processes may contain cycles of activities like a cycle of review that continues till the acceptance of its precedent activity.

3.3 Event based Approaches

3.3.1 Event-driven Process Chains (EPC)

The most commonly used event driven process modeling approach is Event-driven Process Chains (EPC). It was developed as business process modeling language within the framework of Architecture of Integrated Information Systems (ARIS) at the Institute for Information systems (IWi) of the University of Saarland, Germany, in collaboration with SAP AG [Scheer 00]. ARIS framework divides complex business process into four descriptive views and then these views are integrated to form a complete view of the whole business process through EPC. These views are *data view*, *function view*, *organization view* and *resource view*. The transformation to a complete business process is carried out in five phases, called *descriptive levels*, which are characterized by different update cycles. These cycles update the descriptive levels from business problem analysis, requirements specification, design specification, implementation description to information technology realization [Scheer 09].

EPC offers a sequential flow of events and functions to represent the logical dependencies of activities in business process. A metamodel of EPC, presented in figure 3.7, illustrates different process elements used in EPCs [Turan 12]. *Functions* are the active elements of EPCs that model activities and tasks. *Events* serve as the pre and post-conditions for these functions. They describe the circumstances under which a function works and the resulting state of the function. Thus a function can trigger an event. *Process paths* are used to abstract sub processes in a process model. *Control flow* is depicted through arcs that connect events with functions and process paths. A control flow can be split or merged through the use of *logical connectors* of three types: AND, XOR and OR. *Resource unit* models the information/material of the real world. Functions are connected to their input/output data through *information flows*. *Organization units* represent the persons or organizations responsible for a function and are linked to them through *organization unit assignments*.

Once the main constructs of EPC are known, some rules are specified on how to connect these constructs [Scheer 05]. The authors present these rules as:

— Each EPC starts and ends with one or more events.

— An EPC contains at least one activity.

— An EPC can be composed of several EPCs.

— Edges are directed and always connect two elements corresponding to the sequence of activation.

— An event cannot be the predecessor or the successor of another event.

— An activity cannot be the predecessor or the successor of another activity.

— Each event and each activity have only one incoming and/or one outgoing edge.

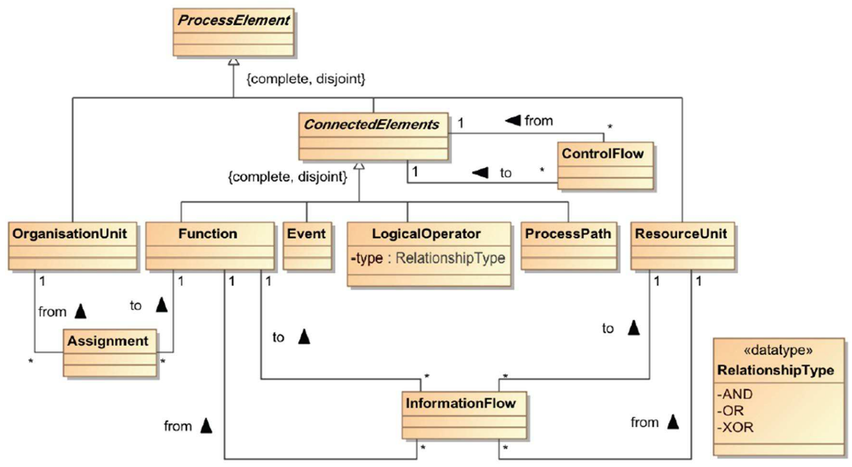


Figure 3.7 – EPC metamodel [Turan 12]

Because of the informal semantics of EPCs, several attempts have been made to present some formal semantics. Formalization of EPCs reduce ambiguity, allow completeness checks and offer model consistency across different vendors. Mendling presents formal syntax and semantics after analyzing different formalizms of EPCs [Mendling 09]. EPCs have been used for defining reference models in the SAP Reference Model, which is one of reasons for it popularity. Apart from this, its simplicity has gained attention from many tool vendors to build its tooling support like SAP R/3 (from SAP AG), ARIS (from Prof. Scheer), LiveModel/Analyst (from Intellicorp Inc.), etc.

Solution criteria based evaluation

EPC is an approach widely used for process modeling in *Business Process Reengineering* (BPR) tools, *Enterprise Resource Planning* (ERP) systems, and *Workflow Management* (WfM) systems. It is a reactive approach, based on events. We evaluate this language based on the chosen solution criteria.

Completeness: EPC offers a minimal set of constructs to develop a process that is simple to understand. These nine basic constructs are further extended by the *extended Event Process-driven Chains* (eEPC). However one feels the deficiency of constructs like goals, intentions, and other constructs related to scheduling and planning. Furthermore the use of events as pre and post conditions for functions is good, but it leaves no room to define further constraints regarding the input and output artifacts of a function. The notion of state for artifacts is not explained. A function can have only one input and output event, which can then be split using control flow. But there is no possibility to model multiple events related to a single functional unit.

Team Development: The notions of roles, actors and organizations are all covered under a single construct of *organization unit*. No support is provided to specify the capabilities of roles. If all these concepts are modeled with the same construct, the model becomes very verbose and difficult to understand both by humans and machines. Apart from this, the choreography of messages between the roles is also not focused in the approach.

Abstraction: *Process paths* are used as a mechanism to abstract sub-processes in an EPC. There is only one entry and exit point for this process path which is very constraining. The use of five different levels of description allows to define the process model at a very coarse level. Then by using update cycles, this level of description is raised to further levels. This helps in refining the process model over time.

Modularity: The concept of modularity is not very well incorporated in the approach, apart from the use of *process paths*, that enable to abstract an EPC within an EPC. Only one event can be used as an entry/exit from it. There is no notion for specification of modules or their interfaces.

Reusability: Reuse of process patterns can be done through multiple usage of process paths in EPCs. However, lack of a concrete modular approach with specified interfaces reduces the possibilities of process reuse.

Tailorability: Functions in an EPC can be replaced with other functions. But a lack of interface specification does not allow to tailor the process in an effective manner. However, event based systems allow a certain level of reactivity. This reactivity allows to create a complex process that can handle various different situations and thus be tailored accordingly by omitting the non relevant parts. ARIS uses multiple descriptive levels for its process models. An EPC can be tailored using the mapping to its abstract descriptive level.

3.3.2 Yet Another Workflow Language (YAWL)

YAWL is a process modeling language that is based on the analysis of existing workflow management systems and workflow languages [van der Aalst 04]. It was developed by Wil van der Aalst (Eindhoven University of Technology, the Netherlands) and Arthur ter Hofstede (Queensland University of Technology, Australia) in 2002. It is developed as an extension to the workflow nets, which are in turn based on Petri nets, a well-established concurrency theory with graphical representation. The execution semantics was originally defined using state machines, but later on a set of Colored Petri Nets (CPN) was used for this purpose [Russel 07].

YAWL allows the development of extended workflow nets (EWF-net) that are hierarchical. The tasks can either be atomic or composite. Composite tasks refer to a unique EWF-net at lower level. A root net is a EWF-net that has not been referenced by any composite task. YAWL allows only one root net in a workflow specification, which is the starting point of the workflow. Each EWF-net consists of tasks (interpreted as transitions) and conditions (interpreted as places). Each net starts with a unique input condition and ends with a unique output condition.

Task and conditions are placed between the input condition and the output condition. They are connected through the edges, thus forming a directed graph. A condition is a waiting state between two or more tasks. An implicit definition of a condition allows to connect two transitions directly. In such a situation, no graphical notation is depicted in the model. A task can have only one entry and exit edge. However a condition can have multiple entries/exits. YAWL uses the logical connector to split or merge the control flow of a model. These logical connectors are: AND, XOR and OR. The semantics of these connectors are the same as in Workflow Nets [van der Aalst 04], except for the OR-join. The semantics of OR-join is defined through a formalism based on Reset nets [Russell 09]. OR-joins impose no restriction on the use of cycles or preceding OR-joins.

YAWL supports tasks with multiple concurrent instances. The number of these instances can be specified at design time and can be dynamically updated during runtime. Tasks can specify four parameters: *lower bound* and *upper bound* for specifying the number of instances, *threshold* to indicate that the task terminates when this threshold of instances has completed, and *static/dynamic* to specify if additional instances can be added after creation. A static task does not allow the addition of new instances during execution of a task. On the other hand, if the task is dynamic, it is possible to add instances while there are some instances currently executing. A task when executed may remove tokens (irrespective of their numbers) from other elements by defining a cancellation region. This adds the possibility to support cancel patterns of workflow. The concept of cancellation region is adapted from the Reset nets [Russell 09]. Reset nets are Petri Nets that offer reset arcs. A reset arc with source place p and target transition t removes all tokens from p upon firing t.

YAWL is implemented in a system consisting of three major components apart from the web servers: YAWL editor, YAWL engine and YAWL custom services [Foundation 10]. The *YAWL editor* is used to develop the process models. The *YAWL engine* is responsible for executing these process models. YAWL engine is implemented using three packages: *Elements* package that contains the elements of the YAWL process model, *State* package that is responsible for storing and processing the state of the process control flow, and *Engine* package that is responsible for running the processes as per the defined control flow of the process models. This engine schedules the tasks and determines the order of task execution. It is also responsible for the data input and output from the tasks. However this engine is not responsible for the execution of atomic tasks. The execution of these tasks is delegated to the third major component of the system, *YAWL custom services*. YAWL custom service is a web based service that receives the tasks from YAWL engine then performs the task activities and finally notifies the engine about task completion, so that the engine can continue its execution for other tasks. Each and every task in a YAWL process model is associated with a custom service at design time.

Solution criteria based evaluation

YAWL is a comprehensive approach based on the analysis of workflow patterns. Workflow patterns for control-flow, resource and exception handling are the main patterns that inspired its development. Our evaluation is based on both the YAWL language and its implementation system, that complements it. A general drawback of distributing the constructs between the core language and tooling system is that the semantics become implementation specific, which is the case with YAWL.

Completeness: Our evaluation for the completeness of YAWL is based on Boehm's WWWWWHH principle [Boehm 96]. YAWL lacks a focus on *Why* aspects as goals, objectives or intentions for each task or the complete EWF-net are not defined. *What, who, how and how much* aspects are not part of the defined process model, however they are dealt by the implementation system through resource and data requirements modeling. The notion of condition between task is explained as a waiting state between the tasks. There is no formal manner to describe the pre and post conditions for the tasks, neither can they be defined for the input and output artifacts. The semantics of YAWL is defined through Petri Nets, which has raised some questions by academia [Börger 12]. These reservations are responded by the founders of YAWL [van der Aalst 12]. For a detailed debate, we refer to the sources [Börger 12, van der Aalst 12].

Team Development: Team development is supported when a process model can be partitioned and integrated easily to be worked upon by different teams. Roles associated to the tasks are responsible for the actual execution of tasks. But for process designers, their ability to develop large process models in teams is affected when the approach is not modular, as in case of YAWL [Börger 12]. The implementation approach does not focus the choreography between the roles as well. This makes the adoption of the approach difficult for *Business2Business*(B2B) environments.

Reusability: The declarative nature of process models in YAWL allows for their opportunistic reuse through process patterns. However, a systematic reuse of process models achieved by following a modular approach and explicit specifications of interfaces is not supported by YAWL. Limited support for refinement in YAWL does not offer the possibilities of reusing process models at different levels of abstraction. The responsibility of execution of tasks in YAWL process models is delegated to the YAWL custom services, which are web services. The use of these web services allows the reuse of executable process models in different service compositions.

Abstraction: YAWL offers a hierarchical process modeling approach, where a composite task refers to a EWF-net. Thus a process contains sub-processes which are abstracted through the composite tasks. Apart from the compositional abstraction, YAWL does not support abstraction of process modeling concepts for the development of process models. It does not offer the possibility to develop multi-level process models like Situational Method Engineering [Gonzalez-Perez 07] or evolution-based process models[Jaccheri 93].

Modularity: YAWL process model has one root EWF-net and can have further EWF-nets if the root net has composite tasks. An EWF-net which is not root net is a module that can be referenced by composite tasks. This composite task has a unique start and end conditions. An EWF-net does not specify its interfaces. It does not encapsulate the process anymore than specifying the start and end conditions. The overall architecture of the process model does not fit into the design by contract approach. However, tasks are executed through web services for their enactment. Use of web services for execution allows modularity but restricts it to the execution phase. This level of modularity is not even supported for all the tasks, because only the atomic tasks can be delegated to web services for execution. Thus execution of the composite tasks as web services that can compose other atomic tasks is not supported.

Tailorability: Each task in YAWL offers a variable to declare it either static or dynamic. In case of a dynamic task, further instances of the tasks can be added to the system during execution. This allows a certain level of tailorability for the process model. Apart from this, YAWL supports exception handling and token cancellations for the process models, which allow the process model to be flexible and support runtime adaptability. But as far as the evolution of process model is concerned, the architecture of the process model lacks modularity and support for abstraction, thus it is hard to evolve the process model.

Enactability: Workflows are supposedly best suited for the enactment, as the process models are handled by process engines that are responsible for enacting the processes. YAWL is inherently based on workflow system and thus offers a concrete process enactment support. It offers a process engine that is responsible for task scheduling. For executing individual atomic tasks, it delegates the responsibility to web services. Thus the tasks are performed by the web services and then reported back to the process engine for the continuation of execution of the process model. Only atomic tasks are mapped to the web services, and no possibility is offered to compose these web services for composite tasks.

3.3.3 Little-JIL

Little-JIL is a visual process modeling language developed by LASER (Laboratory for Advanced Software Engineering Research) of University of Massachusetts [Wise 11]. It has formally defined semantics. It has an associated interpretation framework, Juliette, that supports specification, execution and analysis of processes. It represents processes as a hierarchy of steps carried out by agents. These agents can be humans, software systems or hardware devices. A Little-JIL process model consists of three main aspects: 1) Activity coordination specification using hierarchical decomposition of steps 2) Artifact flow (& control-flow) specification that connects the sub-steps to their parent steps through edges and 3) Resource and agent collection specification needed to perform these steps [Cass 00].

A step (represented as a black bar) is a unit of work assigned to an agent in a Little-JIL process, as depicted in figure 3.8. A step be decomposed into sub-steps, where leaf-steps are the steps that can not be decomposed any further. Each step in the process has a number of badges associated to it. These badges provide the

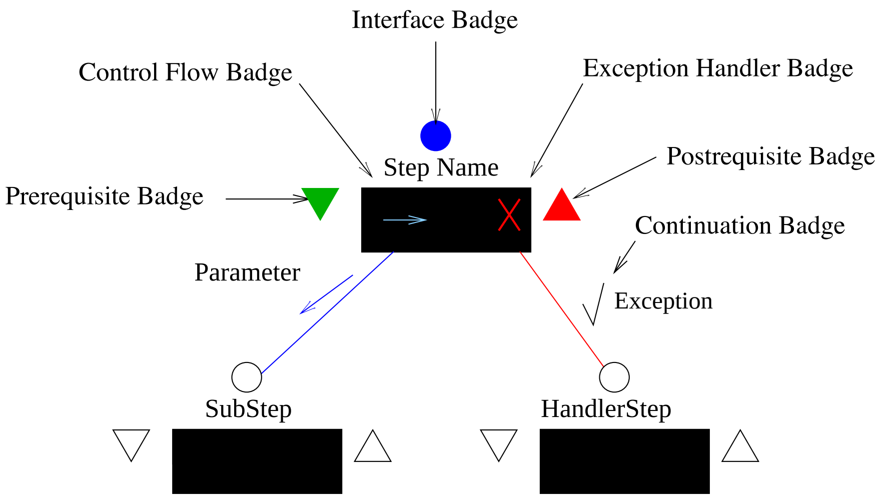


Figure 3.8 – Little-JIL process model legend [Cass 00]

semantic for the step. A circle on top of each step is the *interface* badge that is used to connect the step to its parent step. This interface badge holds the information about the input/output artifacts and the required resources by the step. Artifact-flow is depicted through the blue edges connected a step (from its interface badge) to the parent step. These edges are also used to represent control-flow of the steps. Pre-conditions and post-conditions of a step are specified through the *pre-requisite* and *post-requisite* badges.

Three other badges are associated with a step that are depicted inside the step representation: *Control-flow* badge, *Message* badge and *Exception handler* badge. Control-flow badge specifies the order of execution for the sub-steps of a step. Little-JIL offers four different kinds of control-flow badges: 1) Sequential, where sub-steps execute one after the other from left to right. 2) Parallel, where sub-steps can be executed in any order. 3) Try, where alternative sub-steps are executed from left to right until one of them succeeds. 4) Choice, where agents can choose of the sub-steps to be executed. Other sub-steps are retracted in this case. The leaf-step does not have any control-flow badge and its execution is performed by the assigned agent. *Message* badge (not shown in figure figure 3.8 is represented by a lightening sign in the middle of the step bar. Message handling edges link this badge to other steps or to the environment outside the process to represent the message handling capabilities. *Exception handler* badge is represented by an X sign on the right side of the step bar. Red-colored edges in the model depict the exception edges connect a step to the handlers. These handlers deal with exceptions that may occur during the execution of any of the sub-steps of this activity. An exception thrown by a step is passed up the tree until a matching handler is found to deal with it. Four different continuation semantics are defined by Little-JIL to continue the execution after the exception has been received by the handler: 1) continuing the execution of the step, 2) completing the step, 3) re-throwing the exception and 4) restarting the step. When sub-steps, message handlers or exception handlers are not present, their corresponding badges are not depicted in the step bar. The behavior of a leaf-step depends entirely on its assigned agents, whereas for a non-leaf step, it consists of the behavior of the sub-steps and their order of execution.

Solution criteria based evaluation

Little-JIL is a process modeling language that focuses on the coordination of agents for the execution of business processes. An implementation framework, Juliette is developed alongside Little-JIL, which allows the modeling of the business processes. We evaluate Little-JIL and accompanying implementation framework, based on the identified solution criteria.

Completeness: Little-JIL targets a fair balance of constructs to maintain simplicity and expressiveness at the same time. However, we believe that the process models developed with Little-JIL are a little hard to understand and are verbose. Verbosity in the process model comes from he fact that the control-flow of all the sub-steps are defined through a single *control-flow badge* in the parent step. Thus all the sub-steps can either follow a sequential or parallel (or any other control-flow), but not a combination of them. This forces the process modeler to introduce more steps to adjust the control-flow of the process model. The constructs related to the intents (goals, objectives & intentions) of the steps are not focused. However, these details can be associated to each step through the use of documentations and annotations. The semantics of the process modeling language is precisely defined, which allows process analysis for desirable properties like safety, correctness & reliability *etc.*

Team Development: A *message badge* can be used in each step of Little-JIL to model the choreography between the steps. Each step specifies the messages that the associated agent can send. However, these messages can only be sent when the step is in *started* state. These *messages* correspond to the triggering of (pre-defined) events and the *reactions* correspond to the mechanism for responding to these messages. Support for real-life messages (as in human-human communication) for supporting team development are missing. Steps are allocated to agents, which may be humans, software applications or hardware devices. The concepts of roles or responsibilities associated to the agents is not specified in the Little-JIL model. The distributed development of processes is available through the architectural approach followed by Juliette [Cass 99].

Abstraction: Little-JIL has its roots in a process programming language, JIL [Sutton Jr. 97]. For the development of Little-JIL, the concepts of type declaration mechanisms have been omitted from it. This leaves Little-JIL with a process modeling language that does not take advantage of the typing mechanisms for processes. Little-JIL does not take into account the use of abstractions in its process modeling approach. A single process model has been proposed for all the development phases of the processes. However, use of java in Juliette can be exploited to take advantage of abstraction mechanisms. But, this does not mean that it is part of the proposed language.

Modularity: A notion of module is used in Little-JIL for packaging and reusing the processes. A Little-JIL module may contain steps that are exported (specified through an export arrow in step bar). These *exported steps* are then available to other modules that can reuse these steps by the notion of *imported steps*. Imported steps are not defined inside a module and are reused from other modules that export them. This kind of modularity is good for step sharing between multiple processes. However, a module in Little-JIL does not specify the interfaces for the required and provided services/artifacts. This kind of modularization does not guarantee the replacement of one module with another through the use of standard interfaces. The notion of hierarchy is focused in Little-JIL. Similarly the lack of interfaces does not permit to visualize the inputs/outputs of a high level step without analyzing each of its sub-step individually.

Reusability: The concept of modules in Little-JIL is supposed to support process reuse [Wise 98]. This mechanism is suited well to support reuse of steps within a process model by sharing them through export/import steps. But reusing a step/process between different process models is not that trivial. The approach itself does not offer the use of interfaces or encapsulation. So reusing processes between multiple process models mostly follows an opportunistic style. The process architecture itself is not built around the concept of 'design for reuse'.

Tailorability: The use of events (messages & reactions) and exception handlers allows to develop a reactive process model. However, the concept of *reaction* is not central to the approach *i.e.* all the interactions between the steps is not carried out through the use of reactions. Exception handling capabilities of Little-JIL allow to choose appropriate behavior of the process model in exceptional situations. Little-JIL does not provide any support for tailoring a process model for its customizations. Tailoring a process model that is not executing is more or less opportunistic and no support is provided for guaranteeing the compliance to a specific standard. Tailoring a leaf-step changes the behavior of the complete process, as lack of encapsulation does not allow to separate internal/external behavior. Runtime adaptations of process are not focused in Little-JIL, apart from the inherent reactivity of the process model.

Enactability: Execution support provided for the Little-JIL process models is the key focus area, where it shines. The implementation framework developed along with it, Juliette, is a java-based runtime environment that is responsible for executing the process models. Little-JIL processes are assigned to autonomous agents (human and non-human) that required to report back the success and failure of each step assigned to them during execution. The runtime environment is responsible to coordinate between the agents for performing the assigned work. The use of *channels* allows the communication between the potentially parallel threads of execution. A default lifecycle is associated to each instance of a step with five states: *posted*, *retracted*, *started*, *completed* and *terminated*. Optional steps have a sixth state, *opted-out*. This approach does not allow to developed custom life-cycles for the steps involved in a process. Similarly, there is no support for defining the lifecycle of an artifact.

3.4 Software Process Standards

Different standards are presented in this section to analyze the take of industry for the definition of software development processes. We have focused on the international standards by ISO and IEEE for generic cases. Software development standards from ECSS present a specific software development approach for the development of critical software systems. Most of the standards discussed in this section do not focus on the 'process' of developing software development processes, rather they offer a standard reference process model for engineering a software. Conformance to such a reference process model ensures agreement between different enterprises for *BusinessToBusiness* (B2B) processes. Conformance to such standards are also required for acquiring various certifications for the software enterprises. The intent of presenting these software process standards is to highlight the generic requirements of a software process model. Development of a software process modeling approach should take into account the architecture, structure and a generic behavior of software development processes. IEEE-1074 [IEEE 06] is the only standard discussed in this section that focuses on the development of processes for engineering a software.

3.4.1 International Organization for Standardization & IEEE

In order to standardize the lifecycle processes for both systems and software, two international standards were conceived by International Organization for Standardization(ISO): ISO-12207 [ISO/IEC 08a] and ISO-15288 [ISO/IEC 08b]. ISO-15288 focuses on the lifecycle processes for management and engineering of systems that are made up of hardware, software and humans combined. On the other hand ISO-12207 covers the system aspects, but focuses mainly on the software implementation processes. The important concepts covered by both these standards regarding the system aspects are hierarchical composition of systems, defined system boundaries and interactions between system components generating properties at the boundaries. They categorize the system lifecycle processes in four main categories: *agreement* processes, *organizational project-enabling* processes, *project* processes and *technical* processes. The three later kinds of processes are internal to enterprises, whereas the agreement processes standardize the interactions between different enterprises. ISO-12207 further categorizes the software specific processes in three kind: 1) *software implementation* processes that define the core activities of software development like requirements specification, analysis, design, construction, integration *etc.*, 2) *software support* processes that define supporting activities like documentation, configuration management, quality assurance, verification, auditing *etc.* and 3) *software reuse* processes that focus on reuse asset management and program management.

IEEE-1517 [IEEE 10] is a software standard presented by IEEE that is based on ISO-12207 to describe the system and software reuse processes. It draws a relationship between the systems processes and the software specific processes described in ISO-12207. It promotes the software development process, referred by standard as *application engineering*, to be based upon the usage of assets. These assets are designed to be used in multiple contexts. Reuse assets include the domain architecture and other assets developed for the domain of the software product. The reuse assets may be obtained from other domain engineering processes, organization's reuse libraries, suppliers *etc.* IEEE-1517 presents the standardized definitions for integrating reuse assets in system lifecycle processes and software-specific lifecycle processes.

Previously described ISO and IEEE standards focus on providing a standard set of processes involved in system and software management and development, which might serve as a reference process models. Whereas IEEE-1074 focuses on the process of creating such process models [IEEE 06]. It aims at providing "*a process for creating a software project life cycle process (SPLCP). It is primarily directed at the process architect for a given software project*" [IEEE 06]. It offers a systematic approach for the development of software project lifecycle processes in five distinct phases. The first phase focuses on establishing the requirements for the processes. Once the requirements are established, a software lifecycle model is to be selected. IEEE-1074 does not enforce the use of any specific lifecycle model. The third phase is to develop the software project lifecycle processes. Development of these processes means to arrange their activities in executable sequence based on scheduling constraints. This ordering of activities also takes into account the entry and exit criteria of each activity of these processes. These processes need to be mapped to the software lifecycle model. The fourth phase establishes the software project lifecycle processes by linking them to organizational process assets. The output information of the activities is assigned to the associated documents. Finally the last phase validates these processes.

This study of ISO & IEEE standards for the development of software project lifecycle processes shows that they develop these processes in different phases. Each phase is a refinement of the previous phase, that refines the processes from their initial requirements specification to their validation. These standards do not take into account the enactment of the processes. Besides stepwise development of the processes, they also focus on the hierarchical composition of the processes. They promote the definition of a clear boundary for the processes, where inputs and outputs are clearly specified. For example IEEE-1074 presents an activity having clear boundaries and specified inputs and output, where the input of the current activity can be traced back to the output of the source activity (as shown in figure 3.9). Although these standards do not impose any specific process architecture, a contract based design seems to be an appropriate choice for clear boundary definitions and dependable process component interactions.

3.4.2 European Cooperation for Space Standardization (ECSS)

European Cooperation for Space Standardization (ECSS) is working to provide standardization for the European space sector activities. It targets the cooperation between the space agencies and industry to achieve a consensus over process understandings. Out of the standards produced by ECSS, we are particularly interested in the ECSS-E-ST-40C [ESA-ESTEC 09], which standardizes the processes and activities related to software development for space engineering projects. Many other

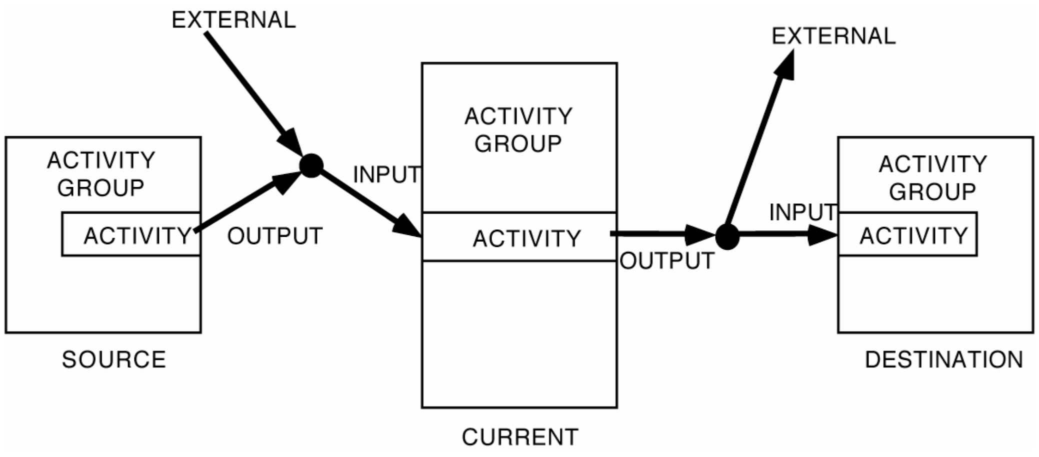
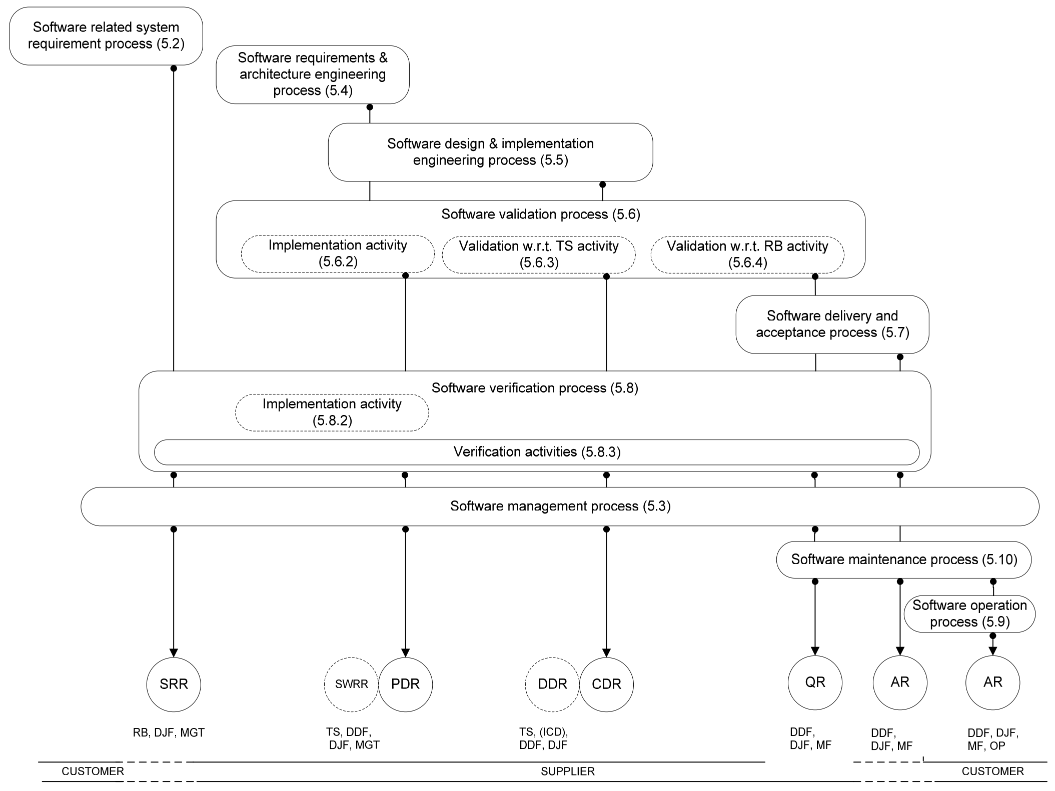


Figure 3.9 – Information flow [IEEE 06]

ECSS standards are referred from this standard targeting standardized vocabulary, product assurance and project management *etc.*

This standard promotes the idea of 'standard for making standards', in a way that permits a supplier to develop his own standard that complies to the requirements of ECSS-E-40 [Jones 02]. It follows a customer-supplier concept, where ESA is typically a top level customer. The suppliers of ESA can further become customers and follow the same standard for subcontracting the software development projects. This creates a chain of customer-supplier relationships extending downwards to the lower levels of subcontractors. Reviews serve as the main interaction points between the customers and the suppliers. This standard can be used to guide the agreement between the stakeholders. This customer - supplier perspective of the standard is depicted in figure 3.10, where the sub-processes are depicted as a choreography between them. Here a source is represented as a supplier and the current organization as the client, which change role in the next interaction, where the destination becomes the client. An organization can establish its processes according to this standard as a dedicated organizational standard. These processes can be supported by a set of methods, techniques, tools and personnel. In such cases organizational standard can be used to establish such environment and ECSS-E-40 to assess its conformity. ECSS-E-40 offers two levels of conformity: *full conformance*, for which all requirements of the declared set of processes need to be satisfied and *tailored conformance*, where a subset of the tailored requirements of the set of declared processes are satisfied and the outcomes are presented as evidence.

ECSS standards do not offer a methodology of software process modeling, so they should not be taken as an approach that can guide the process development approach in terms of technique. Instead, they can be taken as specific software standards which standardize the software development activities. The intention of explaining this standard in state of the art is to explain the general requirements of a process modeling approach. In order to develop a software process modeling approach, it is worthwhile to understand the current requirements of the software industry in



Software related system requirement process (5.2)
Software requirements & architecture engineering process (5.4)
Software design & implementation engineering process (5.5)
Software validation process (5.6)
Implementation activity (5.6.2)
Validation w.r.t. TS activity (5.6.3)
Validation w.r.t. RB activity (5.6.4)
Software delivery and acceptance process (5.7)
Software verification process (5.8)
Implementation activity (5.8.2)
Verification activities (5.8.3)
Software management process (5.3)
Software maintenance process (5.10)
Software operation process (5.9)
SRR
SWRR
PDR
DDR
CDR
QR
AR
AR
RB, DJF, MGT
TS, DDF, DJF, MGT
TS, (ICD), DDF, DJF
DDF, DJF, MF
DDF, DJF, MF
DDF, DJF, MF, OP
CUSTOMER
SUPPLIER
CUSTOMER

Figure 3.10 – Software life cycle processes [ESA-ESTEC 09]

a generic way. For example the hierarchical definition of processes defined in this standard, requires a process modeling approach to be able to model the processes accordingly.

ECSS-E-40C is based on ISO 12207 [ISO/IEC 08a] and defines a set of processes. The requirements on these processes are described as individual requirements for each component activity contained within them. Expected inputs and outputs of each activity is also described. The software development process standardized through ECSS-E-ST-40C has 9 main sub processes. Each of these processes has multiple sub-processes. Figure 3.10 presents the main processes, that are:

— **5.2 -** Software related system requirement

— **5.3 -** Software management process

— **5.4 -** Software requirements and architecture engineering process

— **5.5 -** Software design and implementation engineering process

— **5.6 -** Software validation process

— **5.7 -** Software delivery and acceptance process

— **5.8 -** Software verification

— **5.9 -** Software operation process

— **5.10 -** Software maintenance process

This ECSS standard defines the sequencing and dependencies of the processes, where no particular life-cycle model is imposed, but its selection is an essential management activity. This choice needs to be documented in the Software Development Management Plan. ECSS-E-40C organizes the processes and their activities and tasks in a sequential manner. However it is explicitly stated that, "this positional sequence does not prescribe or dictate any time-dependent sequence" [ESA-ESTEC 09]. It even encourages the use of iteration within activities to offset the effects of the implied sequences.

3.5 Critical summary of approaches

This chapter presents the state of the art in software process modeling languages. As software industry relies a lot on the generic business process modeling, we have discussed those approaches as well. Some researchers categorize these software process modeling approaches in three groups: Process-centric Software Engineering Environments, Business Process Modeling and Workflow Management. However in order to present our thesis, we have grouped them into two categories: flow based languages and event based languages. Languages like BPMN do offer the notion of events, but it is not its central notion for process interactions, so they are placed under flow based languages. Activities in BPMN are arranged in a flow based manner, where they can choose to use the notion of event.

Table 3.1 summarizes the evaluation of state of the art based on the solution criteria identified at the beginning of this thesis. The first criterion is regarding the completeness of a process modeling language. Generally, all the approaches provide enough constructs to model the process. Due to the limited scope of most of the approaches in terms of support for process lifecycle phases, they provide the constructs related to the supported phases only. For example, SPEM does not provide the concept of state for an activity, because it does not support the execution of the processes. Constructs for describing the intentions, goals and objectives of the activities is fairly provided by most of the approaches. Finally, most of the approaches except SPEM have a formal semantics, provided either by the approach itself or by other researchers in academia.

Team development is the resource view of the process model that also takes into consideration the distributed development. All the approaches provide a basic level of task allocations, except BPEL that has issues with allocating tasks to human resources. Responsibility assignment is the definition of privileges associated to the roles for carrying out the tasks. None of these approaches provides a mechanism where responsibilities, roles, actors and their capabilities are all described separately and in detail. For distributed development of process models (including distributed executions) most of approaches provide some mechanisms, except for SPEM and its extensions.

Criteria	SPEM	xSPEM	MODAL	BPMN	BPEL	EPCs	YAWL	Little-JIL
Completeness								
Architectural constructs	+/-	+	+	+/-	+/-	+/-	+	+
Process intents	+/-	+/-	+	+	+/-	-	+/-	+/-
Process behavior	-	+	+	+/-	+	+/-	+	+
Team Development								
Team communications	-	-	-	+	+	-	-	-
Task allocation	+	+	+	+	+/-	+	+	+
Responsibility assignment	-	-	-	-	-	-	-	-
Distributed process development	-	-	-	+/-	+	+	+	+
Reusability								
Approach-based systematic	+/-	+/-	+	-	+/-	-	-	+/-
Implementation-based systematic	+/-	+/-	+/-	+/-	+	+/-	+/-	+/-
Opportunistic	+	+	+	+	+	+	+	+
Abstraction								
Phase-wise refinement	-	-	-	-	-	+/-	-	-
Typing/conformance mechanisms	-	-	-	-	+/-	-	-	-
Modularity								
Hierarchical modularity	+	+	+	+	+/-	+	+	+/-
Contextual modularity	+/-	+/-	+	+/-	+	-	-	+/-
Tailorability								
Static process tailoring	+	+	+	+	+/-	+/-	+/-	+/-
Dynamic adaptations	-	-	-	-	+	+/-	+/-	-
Enactability								
Execution support	-	+	+	+/-	+	-	+	+
Activity lifecycle	-	+/-	+/-	+/-	+/-	-	+/-	+/-
Artifact lifecycle	-	-	+/-	-	-	-	-	-

Table 3.1 – Evaluation of existing approaches based on the solution criteria

Reusability is an important aspect in process modeling. An opportunistic reuse of process models is inherently available in all approaches that have digital models for processes. However, a systematic reuse of the process is not well-supported by these approaches. Many of these approaches provide an implementation based reusability where the implementation tools have some provisions to reuse the process components. But the inherent support for reuse, through a process architecture that follows "design for reuse" is not present in most of these approaches. Other mechanisms of reuse like activity sharing (like export/import step in Little-JIL), or activity pointers (like x-Use concepts in SPEM) are used to support activity reuse. These concepts allow for the process reuse capabilities to a certain level.

The use of abstraction in process models is generally not supported by most of the process modeling languages. This concept is present in PSEE as an implementation support, but again it is not a methodological part of the modeling approach. Abstraction in process modeling approaches can be exploited in two ways. First, by using the refinement concepts over the process development phases. Second, by introducing the typing or conformance mechanisms in process modeling approaches. Typing mechanisms may allow to create process types, process sub-types and then process instances. The use of modularity is present in these process modeling approaches for the representing process hierarchy. Processes are hierarchical in nature and thus every approach has to offer support for hierarchical modeling to a certain level. For contextual modularity, where multiple modules are present in the same context, no approach other than MODAL & BPEL present the concept of interfaces and encapsulation.

Tailorability of process models is mostly supported for the process models when they are not in execution state. However, support for runtime adaptations in scarce in these approaches. Only BPEL provides a concrete support through the use of web-services where individual web-services can be replaced during the execution of the process model. Support for process execution is not provided by SPEM, EPCs and partially provided by BPMN through transformations. All other approaches provide a support for process executions. However, these approaches either do not provide the possibility to define activity life-cycles or offer a hard-coded lifecycle that can not be customized. None of the given approaches allow the possibility to define a life-cycle for the artifacts, except MODAL that provides a very basic, non-modifiable lifecycle.

3.6 Discussion

A general drawback of using flow based process modeling language is the lack of dynamism and reactivity in the process model. However, using the notion of events and exception handling, BPMN offers to develop reactive process models. On the other hand, SPEM does not take this notion in account thus leaving behind the modelers with proactive approach. Some of the extensions to SPEM, like xSPEM, add this capability and the process models are enriched with reactive controls. On the contrary, events are a central notion for interactions between the activities in event based approaches. Two activities in EPCs need to interact through the use of events. Workflow management systems also place the notion of events at the core of the interaction model, which results in the overall reactivity of the process models.

The inherent nature of a process is hierarchical *i.e.* a process is made up of sub-processes or activities. Current process models use the process architectures that model these processes accordingly. However, the modularity of these processes is not the focal concept for many of these approaches. SPEM offers the notion of process component, but the interfaces are restricted to Workproducts. The notion of a process is very close to a service, where each process component provides a service if certain pre-conditions are met. A proper encapsulation of the process, should restrict all interactions including control flow and choreography through specified interfaces. MODAL improves the concept of process component offered by SPEM through the notion of ports that offer services. EPCs and workflow systems do not offer any specific approach for process encapsulation.

Lack of a modular approach in process modeling methodologies limits the reuse of modeled processes. An opportunistic reuse of processes in a process modeling approach is not a very elegant solution to process reuse. Systematic reuse of process models is only possible when the processes are designed for reuse. Many process modeling applications that follow the current approaches, offer the possibility of storing the processes in process repositories [Elias 12]. These solutions offer different functionalities for storing, searching and managing versions. Some of the researchers have even gone as far as process mining to extract information from the processes [van der Aalst 07]. But finding an appropriate process for reuse is one part of the effort and integrating it into a process model is another. Process modeling approaches do not offer appropriate means to integrate a reusable process in a process model. A modular approach that is designed for reuse can help in solving these integration issues. An interesting approach for effective reuse of the processes is presented by the PBOOL+ process modeling language, which models each process as a component [Thu 05]. This language is presented in the RHODES software process modeling environment. One of the choices made by this approach is to use the specification and implementation of elementary components within the complex component. We believe that specification and implementation of a process component are temporal phenomenons, which are introduced to a process model after refinement. This approach takes activities, roles, products and strategies as components. Having four different types of components with different semantics in a single model complicates it as well. We think that a process modeling approach should focus on the componentization of activities, where roles and products should be dealt through the interfaces.

IEEE-1074 [IEEE 06] is a standard that focuses on the process of developing software development processes. It presents the approach in different phases of development from requirements specification till validation. Software process lifecycles are very close to the software development lifecycles. The development of software processes follows the same phases of development like software. Process-centric software engineering environments take these notions into account and offer the possibility to manage the 'step-wise' development of software processes. However, due to the complexity of process programs, they are not adopted by the industry. Other software modeling approaches do not target the use of abstraction in this context. Software process modeling methodologies either focus on a specific phase of development (like BPEL for execution) or tend to model multiple phases in a single model (like BPMN for process specification, analysis, development, validation, *etc.*). An interesting approach is to take advantage of abstraction to model the processes. Similar methodology is also adopted by a process patterns approach [Tran 07]. This approach presents a MOF-based process metamodel inspired by SPEM, that distinguishes between three process patterns: *AbstractProcessPattern*, *GeneralProcessPattern* and *ConcreteProcessPattern*. These patterns capture generic recurring structures for modeling processes, then refine them to partially specified patterns and finally, capture a completly specified solution. We believe that this approach advances in the right direction for taking advantage of refinement, for process modeling. This approach is further complemented by automatic reuse operators that enable automatic applications of process patterns for the generation of process models [Tran 11]. We believe that inspirations from these approaches can result in a process modeling methodology that focuses on the reuse of process elements from different abstraction levels.

A process model developed using any of the current process modeling approaches can comply with a process standard. But the process modeling approaches themselves do not offer any possibility to verify the compliance or offer support for process assessment. For example, an enterprise can develop a process using BPMN where it follows the ECSS standard. BPMN does not offer any support for tracing the process elements back to the standard. The possibility of mapping process elements to one or multiple process standards is of vital importance for software enterprises. Software enterprises go for continuous process improvements. Tailoring the processes for specific projects is a common practice. Verification of process conformance against adopted standards is important in such scenarios. Workflow management systems offer the possibility to verify the conformance of executing processes against process models [van der Aalst 05a]. Other mechanisms offer the possibility to verify the compliance of the executing software processes to initially declared process objectives [Sadiq 07]. These notions are different from verifying the conformance of a process model to a process standard. Possibility to model the compliance to process standards is important for processes in different phases of development, even dynamic adaptations of processes during execution should not breach the standard, especially in a domain where certifications are crucial.

Some current software meta-modeling approaches like power type based modeling [Gonzalez-Perez 06], deep instantiation [Atkinson 02] and lazy initialization (LIMM) [Golra 11] are offering the support for multi-level modeling. A recent international standard for development methodologies, ISO-24744 [ISO/IEC 08c] is based on the power type based modeling concepts. These trends in the domain need to be tackled by the process modeling approaches by offering the possibilities of multi-level modeling. Current process modeling approaches do not offer any mechanism to model the processes in multi-level environments.

Part II

Process Modeling Framework

Chapter 4

Structure of Metamodels

Contents

Abstract - *This chapter presents the structure of the proposed process modeling approach, Component-oriented Process Modeling Framework (CPMF). It starts with giving a holistic view of the approach and then presents each metamodel used in this approach separately. Each metamodel is described using a common scenario, as an example. The interaction mechanisms within the process architecture of the proposed modeling framework is outlined. Finally, the refinement of contracts for each activity as it passes from one model to another is described.*

4.1 Multi-metamodel Process Framework

4.1.1 Component-oriented Process Modeling Framework

Modeling has become a central part of software development lifecycle. It not only guides the development but also becomes a part of the system. The design decisions taken for the development of the software are carried out till the realization of the final system. These design decisions collectively take the form of various models in different phases of development, till the concrete code. Each step in this progression is modeled using constructs, whose definition is sought in given metamodels. This remains a designer choice to move from one model to another manually or using semi-automatic transformations. Having a concrete family of metamodels is very important, so as to realize the objective of Model Driven Engineering. Software development through the evolution of models from requirements till deployment passing through a series of transformations is the hallmark of MDE. These transformations are responsible for creating, modifying, translating or refining models, as the software development project advances. A family of metamodels gives a sound base upon which the multi metamodel application development can be realized.

As far as the current process modeling approaches are concerned, they rely on a unique metamodel for the development of process models. SPEM [OMG 08], xSPEM [Bendraou 07], BPMN [OMG 11], BPEL [OASIS 07] and YAWL [van der Aalst 04] are all based on fairly complete metamodels spanning over multiple packages. Each of these approaches models the process either in a single phase of development (*e.g.* BPEL for enactment phase only) or in multiple phases (*e.g.* BPMN for requirements specification, analysis, development, validation *etc.*). We argue that capturing the semantics of the processes at different development phases is difficult using a unique process model. For example, the focus of an enterprise on a process is different in requirements definition phase than in implementation phase. Each phase of process development, just like software development, needs specific level of details, tool support, management decisions, *etc.*

> **Contribution I:** *Specific language for modeling the processes in a particular phase of process development lifecycle, that refines with each passing phase.*

For a clear and precise modeling support of a process in a specific phase of development, we propose to use a specific metamodel. This means that for each phase of process development we have a distinct metamodel. This would result in a family of process metamodels that deal with modeling the same process in its different phases of development. The number of metamodels depends on the chosen *process development lifecycle*. We do not enforce to use any specific process development lifecycle. The idea of multi-metamodel development is based on a simple rule, "particular process metamodels for particular process development phases". For the purpose of illustration of our approach, we have chosen four phases: *process specification*, *process implementation*, *process instantiation*, *process monitoring*, as shown in figure 4.1. The monitoring phase is not a development phase, so we do not need a metamodel for that phase. The users of this approach may choose more phases and develop the respective metamodels. For now, we present our approach through three metamodels, each for the specified phase of process development *i.e. Process Specification Metamodel*, *Process Implementation Metamodel* and *Process Instantiation Metamodel*. An execution model is generated once the process instantiation model is interpreted by the process interpreter. This allows to execute the processes and eventually monitor them.

> **Contribution II:** *Offering a set of metamodels that defines the process modeling language for three different phases: specification, implementation and instantiation.*

Figure 4.2 shows the approach in two levels of modeling hierarchy. In the metamodeling layer, we see the three metamodels and the refinement relationships between them. We believe that each phase of development refines the process by adding more detail to it. In the lower level, respective process models are depicted going through a chain of model transformations. Each transformation refines the process model and

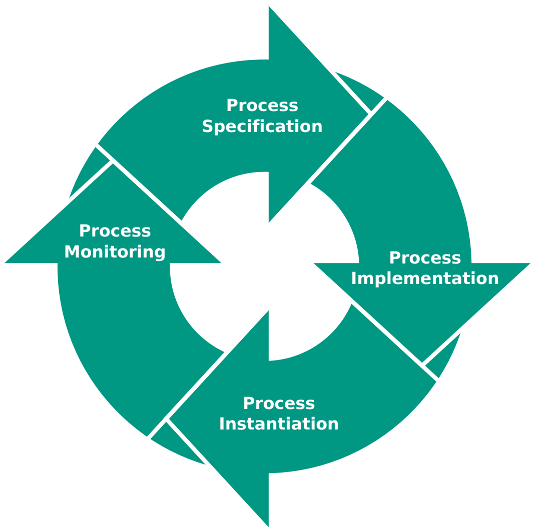


Figure 4.1 – Process Metamodels for Multi-metamodel Development

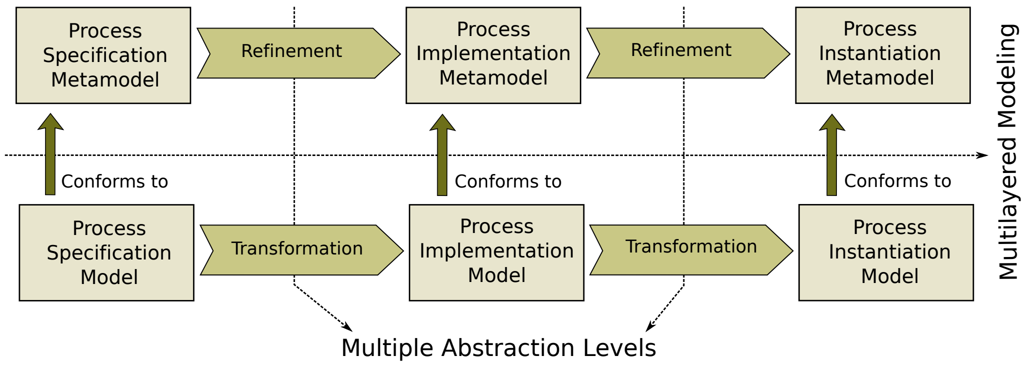

Process Specification Metamodel
Refinement
Process Implementation Metamodel
Refinement
Process Instantiation Metamodel
Conforms to
Conforms to
Conforms to
Process Specification Model
Transformation
Process Implementation Model
Transformation
Process Instantiation Model
Multilayered Modeling
Multiple Abstraction Levels

Figure 4.2 – Process Metamodels for Multi-metamodel Development

adds new slots in the model, to inject further details and design choices. This is an abstract level diagram, which does not show the injection of details. We will discuss that later with refinement of each level in section 4.2.

The *process specification model* (PSpec) is developed in conformance to the *Process Specification Metamodel*. Such a model is based on the requirements specification phase of process development. A software process is specified using this model and therefore is not overloaded with implementation level information. It may be used to document the process best practices in terms of their structure. It is not specific to any organization or project. This added level of abstraction in terms of specification promotes re-usability of the process models. Process standards and best practices are documented in a reusable manner, where they can further be applied to any specific project or organization in multiple ways. The *process implementation model* (PImp) conforms to the *Process Implementation Metamodel*. Such a model documents the specific project details, which are incorporated in the model by adding the implementation details of the process model. The *Process Implementation Metamodel* is semantically richer, so as to express all the fine details of the process model in terms of implementation. This way a single process specification model can be used for multiple implementations, in different projects within or across multiple organizations. Finally, *Process Instantiation Model* (PIns) is developed which establishes the processes by connecting them to the development tools, documents, repositories, people, *etc.*

Let us take an example of an organization that develops its models in conformance to the European Cooperation for Space Standardization (ECSS) standards. If their process model is conforming to the ECSS-E-ST-40C Standard [ESA-ESTEC 09], then this information should be evident from the process model. In such a case, this organization can develop a PSpec based entirely on this standard. Or it can choose to tailor the reference process metamodel offered by the standard, and model that as its PSpec. In process implementation phase, this organization would need to refine this model to carry the implementation specific details as well. Thus a specific implementation model for a project conforming to this standard, can be modeled as PImp. Finally, this organization would want to enact these processes. Instantiation level details for the process model are added in the third phase when a PIns is developed. PSpec and PIns could be developed manually, but in the favor of process model automation, we use transformations in between these process metamodels. Adding the transformation in between these metamodels surely does not automate the process, but helps take the first initiative towards a better automation of process modeling.

For this organization in process specification phase, its focus remains on the specification of interfaces based on the artifacts and their types. The designer level decisions for implementing the process are taken in the implementation phase. The decoupling of the activities which had no importance in the first phase, might be beneficial for this phase, if activities are performed by different roles in varying environments. Finally, the enactment of this process demands further support for project management like project planning, risk handling, scheduling, costing *etc*, which is offered in the instantiation phase. As we believe that the semantics of a process model is different at each phase of development and that these differences in semantics (and syntax) should be incorporated semi-automatically in the model, when the phase requires so. We achieve this through the use of distinct process models and the use of model transformations between them. This approach models the processes from specification till their execution in a consistent manner. No mapping to external execution languages is required like in SPEM or BPMN for enacting the processes. Similarly the approach itself provides the facility to specify the processes. Hence, no translations are required between specification languages and process implementation/execution languages like YAWL or Little-JIL.

Contribution III: *Providing a much wider coverage of process lifecycle phases without translation to other languages (to cover the non-supported phases) resulting in semantic consistence.*

4.1.2 Process modeling scenario

Comparison of different approaches for solving a particular problem is of high value in the research domain. Keeping this vision in mind, a standard benchmark software process modeling problem was developed in the 6th International Software Process Workshop (ISPW-6) [Kellner 90]. This benchmark presents a software change process, which focuses on the designing, coding, testing and overall management of this process. It is assumed that a Configuration Control Board (CCB) authorizes this process which aims to change a single code unit. The complete process is encapsulated in a single higher level abstract activity, named as *Develop Change and Test Unit*, here after referred as DCTU. This activity is composed of other sub-activities, each of which has a separate objective. These activities are:

— **SAT -** Schedule and Assign Tasks

— **MP -** Monitor Progress

— **MD -** Modify Design

— **RD -** Review Design

— **MC -** Modify Code

— **MTP -** Modify Test Plans

— **MUTP -** Modify Unit Test Package

— **TU -** Test Unit

The first sub-activity in this benchmark is the SAT activity, which is responsible for developing a schedule and assigning tasks to the concerned roles of other sub-activities. This activity is carried out by the project manager. SAT is the first activity that is executed with the execution of DCTU activity and it starts as soon as it gets the authorization from the *Configuration Control Board* (CCB). Its input contract for *RequirementsChange* and *ProjectPlan* are delegated from the parent activity *i.e.* the *RequirementsChange* document is handed over to this activity directly from CCB in a hand carried transaction at runtime. However it accesses the *ProjectPlan* from a file (computer I/O). The task assignment is emailed to the roles of other activities rather than the activities themselves, which means that those activities are still not executed but their resource allocation has been done, through this activity. It delivers the *RequirementsChange* document to all the assigned personnel for other activities. Finally, the updated project plans is written to the file. This activity ends, once all the outputs have been provided.

The second activity in this process is the MP activity, which is executed in parallel with all other activities till the termination of DCTU. It is performed by the project manager. This activity is responsible for gathering the *completionNotifications* from all other activities. It starts as soon as the SAT activity ends. It has access to the *ProjectPlans* and can alter them if the project is not advancing as planned. In case, the *ProjectPlans* are changed, it notifies it to all the assigned personnel of other activities. The *CancellationRecommendation* for DCTU is verbally initiated by this activity. This activity is also responsible for aborting the DCTU if the *Cancellation-*

Recommendation is approved by the CCB. This means that as parent activity can trigger lifecycle events for child activities, child activities can also trigger the lifecycle events for the parent activities. These events are then propagated upwards/downwards to the desired level. This activity ends once the DCTU is canceled or the TU activity is successfully completed.

MD activity is responsible for modifying the current design, based on the requirements change document handed over to it by SAT activity. A design engineer is responsible for carrying out this activity. It has access to the software design document file to get the *current design*. Once the design is modified, it is hand carried to RD, MC and MUTP activities and this activity ends. However subsequent iterations can begin if RD activity does not approve the design. This activity is followed by the RD activity, that reviews the modified design and provides a feedback to MD activity. A feedback loop between MD and RD illustrates an iteration in a part of the sub-assembly of the process. This iteration continues till the modified design is approved by the RD activity, which notifies the review decision for each iteration to the MP activity. Approved modified design is handed over to the MC and MUTP activities by MD and documented in the software design document file by the RD activity.

A quality assurance engineer performs the MTP activity, as soon as the tasks are assigned by the SAT activity. This activity receives the hand carried, currentTestPlans and modified it according to the *RequirementsChange* document. Once the *modifiedTestPlans* are ready, they are hand carried to MUTP activity. MTP activity terminates once the output is ready. MUTP activity is also performed by a quality assurance engineer, when the MTP activity is terminated. MUTP activity receives the approved *modified design* from the MD activity and (if needed) can access the modified source code from the software development files. It is responsible for the actual modification of the *testUnitPackage* (from test package file) for the associated code unit. The modifications result in a new version of *testUnitPackage*, which is kept under automated configuration management. This activity ends once the outputs have been produced. However subsequent iterations may start, based on the (verbal) recommendations of the TU activity.

In parallel to the MTP activity, MC activity modifies the code according to the approved *modified design*. It is performed by the design engineer. It can access the existing code from the software development files. Once the code is modified, and a clear compilation is achieved without errors, the code (source and object) is documented in the software development files and the activity ends. Subsequent iterations begin as soon as the TU activity notifies some problems with the code. Finally TU activity tests the modified code and emails a notification to the MP activity. This activity is jointly performed by the design engineer and the quality assurance engineers. It accesses the *objectCode* from the software development files and the *testUnitPackage* from the test package file. Test results are documented in the Test history file. A verbal feedback is provided to the MC activity regarding the code and MTP activity regarding the *testUnitPackage*. This feedback may initiate subsequent iterations of MC or MUTP activities. The feedback loop between TU and MC activity continues till all tests are successful. It terminates after sending a successful testing notification to the MP activity.

4.2 Metamodels for Process Development Phases

CPMF defines three metamodels to demonstrate the applicability of the approach on a minimal process development lifecycle. Each metamodel is explained below according to its relevance to the specific phase of process lifecycle.

4.2.1 Specification Phase

The first phase of process development lifecycle is to specify the processes. The development of a process model like any other model can be either prior to the system or after its development. The process models that are built prior to the system are developed to guide the actual real life processes that need to be executed. On the other hand, process models are also built to evaluate existing processes, and for that they are developed by abstracting the executing processes. As we are focusing on the software development processes, thus we take into consideration the process models that are developed to guide the development and management of the future processes. In the specification phase of process development, the key interests are as follows:

— **Review customer requirements and objectives** If the organization is developing the process for its own need, then it would be considering its own requirements.

— **Define the scope of the project** The scope of the project defines the project boundaries. This gives an overall objective to the process model, that needs to be limited.

— **Review the business strategy** An initial process model serves as a basis to review the business strategy. A business strategy does not contain the implementation logic of the processes, rather it is based on the objectives and how these objectives are broken down into sub-objectives to achieve it.

— **Identify process components** Once the business strategy is defined, the identified sub-objectives are associated to different process components. These process components are responsible to execute the primitive level steps (directly or indirectly by delegating the responsibility to sub-process components) to achieve these objective.

— **Identify the artifacts** The objectives of a software development project involve the development of software programs, documentations, standards, models, use cases etc. These artifacts need to be identified and specified in the specification phase of software process development.

— **Define the responsibilities** The process activities/tasks are carried out manually, automatically or semi-automatically. In all these cases, responsibilities need to be defined for each task.

— **Preliminary analysis and evaluation of the process** A process model in a specification phase should allow for a preliminary analysis and evaluation to assess if it can meet the customer requirements.

— **Comparison with industry and international standards** If the customer has already defined its own standard or follows some international standard, the compliance of the process model should be checked against them.

Process Specification Metamodel

Process Specification Metamodel (PSpec) is used to define the basic structure of the process model at the specification phase. A process in its specification phase is decomposed into different process components, which creates a hierarchy. Different process modeling approaches adopt different architectures to incorporate hierarchy in the process model. The definition of process hierarchy in CPMF is a little different than others, for the reasons that they usually offer a direct hierarchy of activities in a process e.g. in SPEM. On the other hand, EPC's use a concept of process path in an EPC that refers to another EPC, thus creating a hierarchy. CPMF is very strict in the definition of a process and an activity. We define a process as , "an architecture of interconnected activities such that they collectively aim to achieve a common goal". An activity is a unit of processing action in a process model. Thus a simple hierarchy of activities only would be a structural breakdown of activities with no interconnection. Activities can either be decomposed further or can represent the primitive level of processing. For a meaningful process hierarchy in CPMF, each composite activity contains a process, which amounts to containing a collection of *interconnected* activities. Activities that are not decomposed in a process specification model do not represent the fact that they can not be decomposed any further, it just specifies the smallest unit of breakdown, which is a designers choice.

> **Contribution IV:** *Offering the constructs related to a specific phase of process development only, without polluting the model with additional noise.*

Apart from offering a simple process hierarchy, CPMF offers activity sharing amongst different processes. An activity can be contained by two different processes, which share some common processing actions. A shared activity may get its inputs from any or both of these containing processes, its output being accessible to both the processes.

Inspired from *Design by Contract* (DbC), all the interactions to and from the component are handled through specified interfaces. An activity behaves as a black box component, where the interface to its context and content (in case of composite activity) is through its *contract*. The term 'contract' is an inspiration from DbC, which uses it to describe the conceptual metaphor with the conditions and obligations of business contracts. A contract in CPMF can either be required or provided, which is called its direction. A required contract is an interface specification of an activity for the input artifacts. A provided contract is an interface specification for the artifact

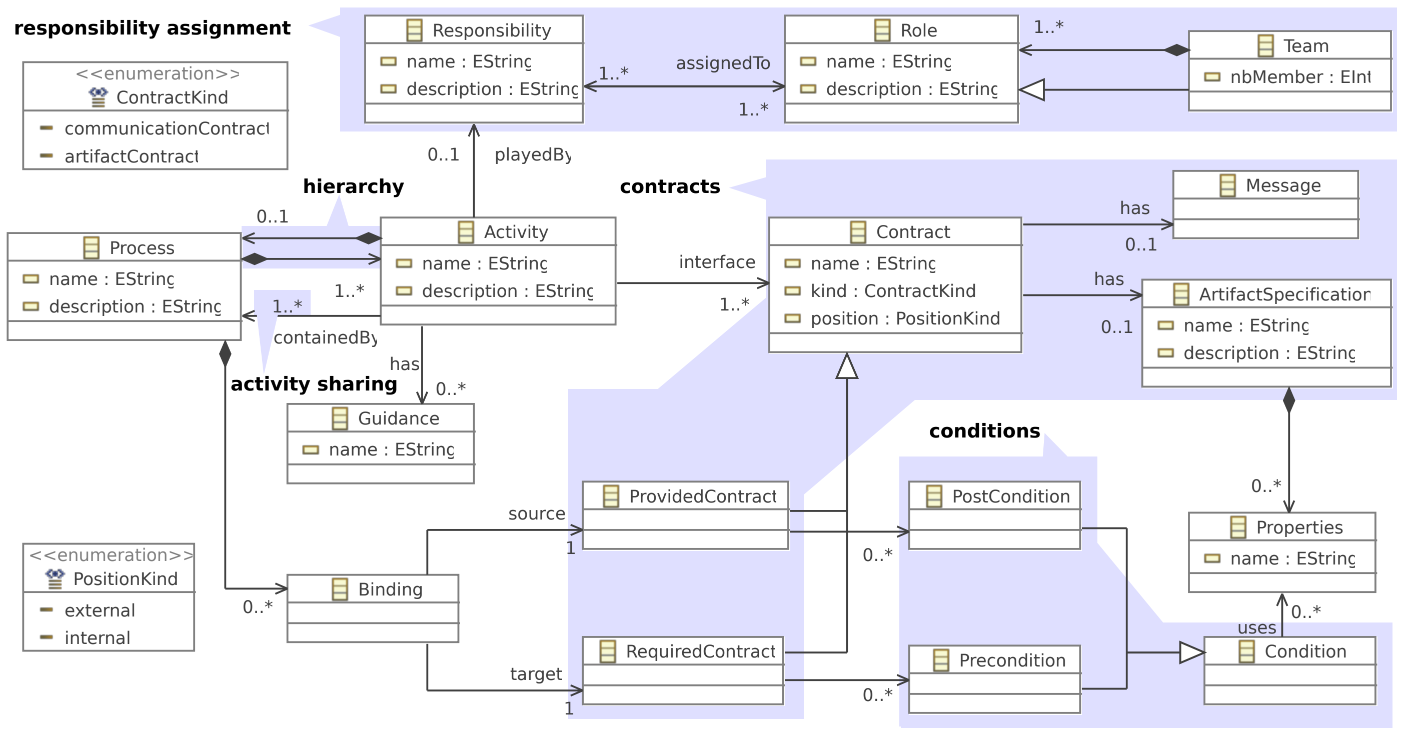


Figure 4.3 – Process Specification Metamodel

that it offers. Besides being required or provided, a contract is either external or internal. External contracts are used to interact with the context of the activity. The context of an activity is the set of parents activities and all other activities composed by the processes that contain it. Internal contracts are used to interact with its content by a composite activity. The content of an activity is the set of all activities contained by the process that it contains. In case of composite activities, for each external contract there is a corresponding internal contract of the opposite direction. Thus for a required external contract, there exists a provided internal contract and vice versa.

> **Contribution V:** *Contractual interactions between the activities guarantee its correctness. Focus on composition and information hiding.*

Each contract of an activity defines an artifact specification. This *artifact specification* specifies the artifact that an activity needs as an input through its required contract or offers as an output through its provided contract. Because the process model is at specification level, the process model is not executable. Thus no actual artifact is passed between the activities, at this level. An artifact specification defines the properties of an artifact. An artifact specification can be optional or mandatory for an activity. This is expressed through the contract, which can either be optional or mandatory. Unavailability of an optional required contract does not create any hindrance for the processing of the activity, whereas a mandatory required contract is obligatory for the processing of an activity. Same ways, a mandatory output contract has to be offered by an activity before its completion, however the optional output contract does not stop the completion of an activity.

In order to guarantee the correct processing of an activity, the contracts of the activity are enriched with conditions. These conditions are based on the properties defined by the artifact specifications. Pre-conditions are defined in the required contract of an activity. They are used to evaluate the properties, that need to be met in order to guarantee the correct inputs for the proper processing of an activity. Post conditions on the other hand are defined in the provided contract of an activity and record the conditions created by the processing of an activity. They guarantee an accurate output of an activity. The contracts of two activities are interconnected through *bindings*. A binding connects the provided contract of an activity to the required contract of another activity, such that the artifact provided by the first activity can meet the preconditions of the second activity. This defines a flow of activities based on their artifacts. This flow of activities is sequential and proactive.

Each activity defines some *responsibilities* for their processing. Each responsibility is assigned to a role or a team. There are different mechanisms for representing role assignments. For example RACI method proposes the RACI matrices for describing the responsibilities assigned to different roles and teams [Hallows 02]. RACI stands for (R)esponsible, (A)ccountable, (C)onsulted and (I)nformed. SPEM 2.0 uses the RACI-VS method to construct the responsibility assignment map that can be attached to the activities, processes and work products [OMG 08]. CPMF does not restrict the use of any specific responsibility assignment approaches, but offers the mechanism to assign responsibilities to the roles. A role defines the function assumed by a person or tool for executing a process. A team is a collection of roles that assume a collective function.

Process specification model of CPMF can be translated to the other existing process modeling approaches. However, this translation would result in a conceptual loss. There is no mechanism to differentiate between an artifact and an artifact specification, both in BPMN, SPEM or other approaches discussed in the state of the art. Similarly a concept of responsibility assignment to an extent of defining the privileges associated with each role is also missing. A translation in the opposite direction, from BPMN or SPEM would also result in considerable loss of data. The reason for this loss is that those approaches mostly target the implementation phase of process development, whereas this process model is used to capture the specification details only.

ISPW-Scenario in Specification phase

We present an implementation of the ISPW process modeling scenario presented in section 4.1.2, as shown in figure 4.4. The scenario presents a *DevelopChange-AndTestUnit* (DCTU) activity, which is a composite activity. DCTU contains an internal process, which further composes eight sub-activities. DCTU has the required and provided contracts for handling the inputs and outputs through the activity. As it is a composite activity, for each external contract, it has a corresponding internal contract of different direction. Each of the sub-activities of DCTU have their respective external required and provided contracts. The *binding* between these contracts is explicitly drawn through the edges that connect these contracts. These edges are connected and represent the direction of data-flow. For example, the *ModifyDesign* (MD) activity offers a provided contract for *modifiedDesign* and the *ReviewDesign* (RD) activity has a required contract for *modifiedDesign*. A binding between the two contracts shows that RD activity is expecting the *modifiedDesign* from MD activity. This binding is depicted by the directed edge connecting the two contracts.

The graphical model of this scenario only depicts the activities, their contracts and the bindings between their contracts to show dependencies among the activities. Other information regarding the *guidelines*, *conditions*, *responsibilities* and *roles* is not shown in the graphical model. These details are entered into the model through the *activity inspector* provided in the model editor. Other views of the model can be generated to present other information held in the model, but not shown in this graphical notation. Generation of these views is out of the scope of this thesis. The model contains this information for the development of these views but no views are generated by the tool support as yet.

Specification level model is not polluted with the implementation details of this scenario. Thus, the activities shown in the model are definitions which do not have any associated implementation yet. Logical connectors for the data-flow between the activities are not shown explicitly in the model. They are encoded within the contracts of the activities. All the required contracts are of *OR-type* and all the provided contracts are of *AND-type*. Pre-conditions are used to express any AND-type input

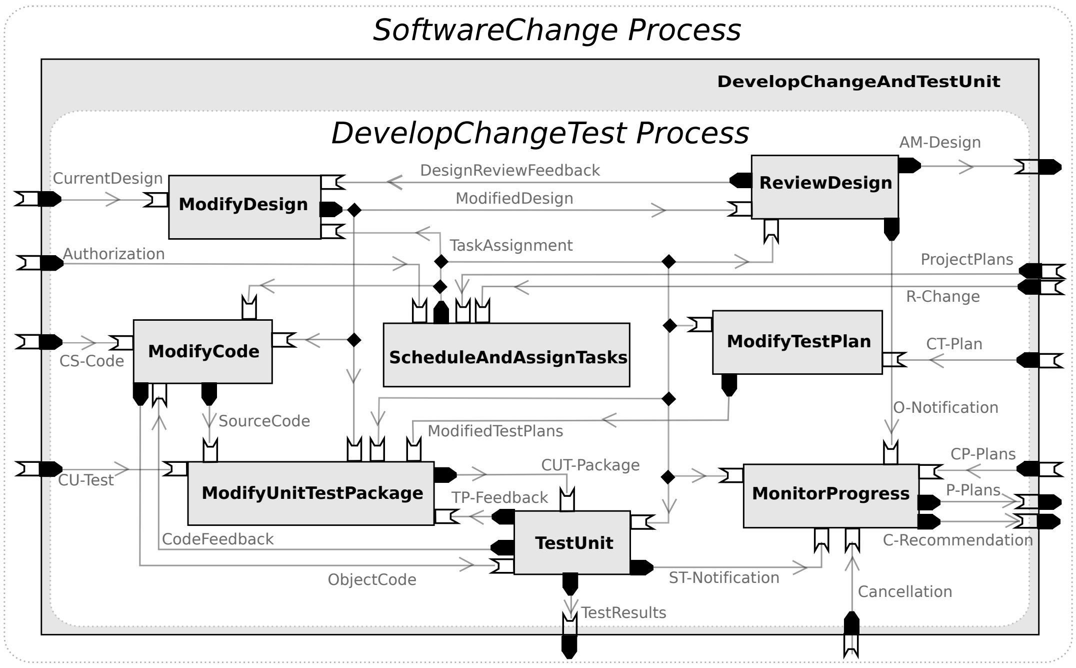


Figure 4.4 – ISPW Scenario - Specification Phase

to an activity at this level. ISPW-scenario describes the artifacts as the inputs and the outputs of the activities. However, the required and provided contracts of this model offer the artifact specifications for respective artifacts. For example, the *ModifyCode* (MC) activity has two required contracts: *modifiedDesign* and *currentSourceCode*. In this case, these required contracts specify what sort of artifact is expected by the MC activity to continue its processing, through presenting the artifact specification. For its provided contracts, it owns two interfaces: *sourceCode* and *objectCode*. These contracts offer the artifact specifications for their respective artifacts.

4.2.2 Implementation phase

Once the processes are specified they provide an initial structure upon which the implementations can be built. Goals and objectives of the processes are already set, but they need further refinement for being project specific. Process specifications are very general in nature and do not take into account any details regarding the specific organization or project. They also do not give the implementation details of the processes. These details are added in the implementation phase of process development lifecycle. The key interests in process implementation phase are:

— **Assess project level conditions** The constraints and conditions specific to a software development project need to be taken care of in this phase of software development. The conditions like the overall duration in which the project needs to be completed, the level of resources available for a specific project are examples of concerns which are specific to a particular development project.

— **Evaluate technical environment** The technical environment for the development of a particular software development project is the level of capability available for its development. Analysis of the kind of infrastructure available to develop a particular process is also done in this phase.

— **Define operational constraints** Once the project level conditions and the technical environment is analyzed, the process can be developed taking into account the external (contextual) constraints. These constraints need to be defined, so as to assist in process development.

— **Analyze & quantify business process** Already available specification level process gives a general analysis capability, but for a particular project it needs to be analyzed for the process fine tuning. For example to analyze a process, we have to identify the requirement of (possible) iterations of an activity. The quantification of process properties sets a limit for process properties that an activity can implement.

— **Identify (alternative) solutions** A specification level process model is comparable to a problem rather than a solution. This problem identifies the inputs and outputs of an activity without precising the solution for implementation. Different solutions for these process 'problems' need to be identified in this phase.

— **Identify & define primitive level work units** Processes are designed down to the primitive level, where the primitive tasks need to be identified. These primitive tasks make up the higher level activities. Implementations need to be defined for the processing of these tasks.

— **Identify process implementation candidates** Once different solutions are at hand, the possible implementation candidates can be chosen, taking into consideration the contextual constraints and the operational constraints. Implementations for the chosen solutions are developed which might result in multiple implementations for a single activity.

— **Define people and system intervention** A software development process needs the intervention of people and system for its implementation. Roles are already associated to activities in the specification level model, but they need to be associated to the process implementations in a way that each role is associated to the task level.

— **Document & store the process implementations** Finally, the implementations of the processes need to be documented and stored in the process repositories. The documentation of these activities allows to (re)use them in the subsequent phases. They are stored in the process repositories for possible retrieval.

Process Implementation Metamodel

Process Implementation Metamodel (PImp) is semantically richer than the *Process Specification Metamodel*. PImp focuses on the separation of concerns, the usage of event based mechanisms and the dynamism introduced through multi-level modeling. *Process Implementation Metamodel* defines the overall structure of the implementation level processes. The metamodel is built using two packages: Core package and the Contract package. The later package is merged in the former to get the complete metamodel, which is presented using UML syntax.

The Core package of PImp defines the core entities of the process model except the details of the contracts, as illustrated in figure 4.5. A process in our framework is a collection of activities that are 'interlinked' to achieve a common goal. This metamodel presents the process model in two levels: abstract and concrete. The abstract level defines the data-flow within a process and the concrete level defines the control-flow. Activities at both levels define their contracts. These contracts allow them to interact with other activities (within their context or through a containment relationship). These activities are not sequenced in terms of a workflow. In fact, their contracts define if they depend on some other activity for their execution. The 'inter-linkage' of activities is explicit on the abstract level, whereas on the concrete level, there is a dependency between the activities that is not explicitly represented. Thus, instead of focusing on a proactive control for the process, we stress to focus on the completeness of definition for each individual activity. Inspirations from the component based paradigm have led us to define an activity as a black box component, where the only interface of an activity to its context or content (in case of composite activity) is through the defined contracts. This gives a nature of pipe & filter architecture to the process, where the activities serve as filters and the dependencies serve as pipes.

> **Contribution VI:** *A bi-layered approach for process modeling that offers separation of concerns for data-flow and control-flow.*

An activity, being the basic building block of the process model, expresses its interactive requirements for operation through contracts. A process containing both activities and their associated dependencies represents an architecture with activities as basic entities and dependencies to define the flow between them. The activities only express their dependencies, and thus the absence of 'hard coded' control-flow introduces a fair amount of dynamism in terms of activity sequencing. This dynamic sequence of flow equips the process model with a reactive control that has the capability of restructuring the control-flow for the activities at runtime. For example, two independent sub-activities planned to be performed one after the other, may be performed in parallel, if the project gets late from the initially planned schedule.

Activities of the process Implementation metamodel follow the same hierarchical structure as followed in the process specification metamodel. There are two parallel hierarchies defined by this metamodel: one at abstract level and the other at the concrete level. At the abstract level, there is a hierarchy of *AbstractProcess* that contains *ActivityDefinition*. However, there is no specialization of primitive or com-

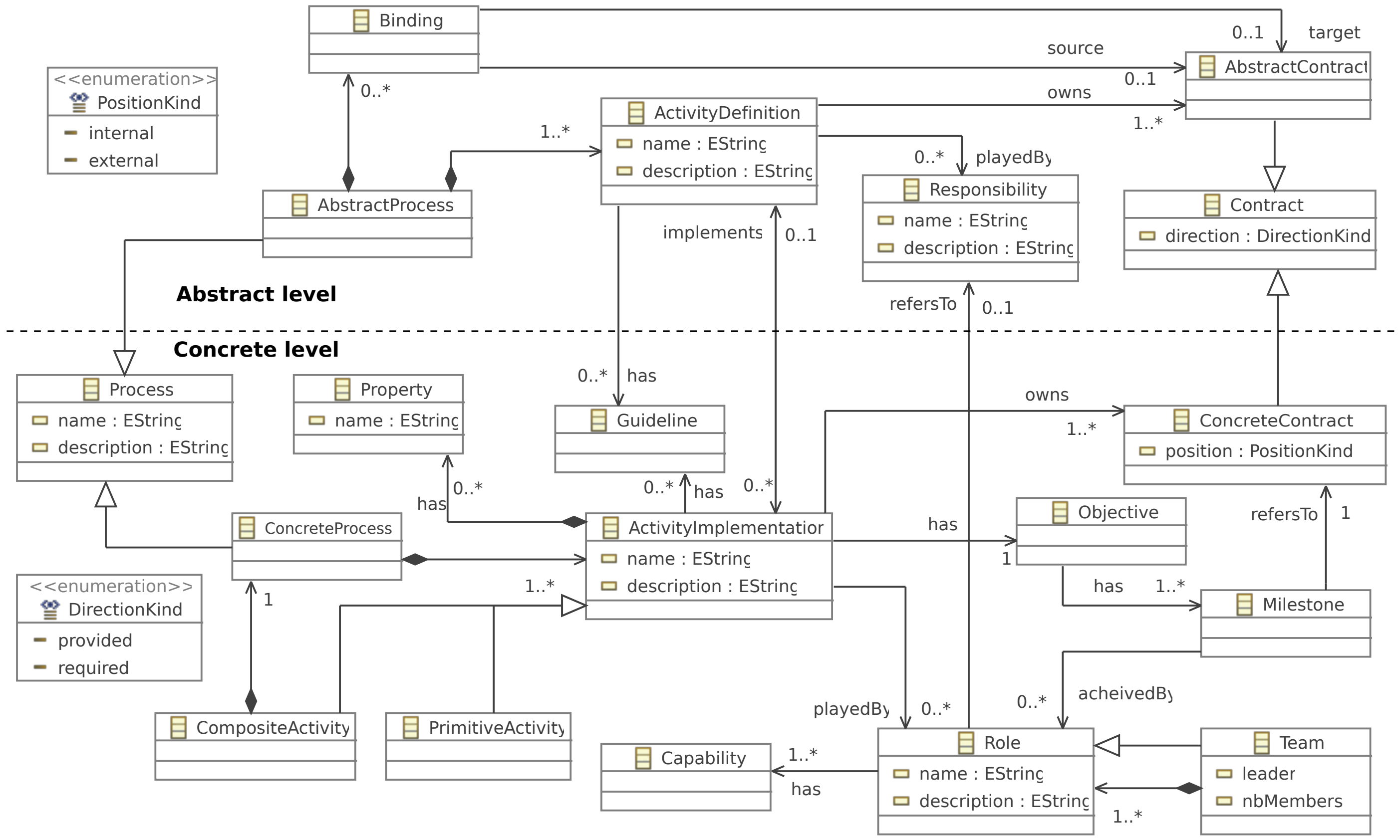


Figure 4.5 – Process Implementation Metamodel - Core Package

posite *activityDefinition*. The implementation of an *activityDefinition* (at concrete level) specifies, if it is implemented through the composition of some other primitive activities or is a primitive activity itself. *ActivityDefinitions* at the abstract level of this model are carried forward from the *Actvities* in specification metamodel.

Considering the importance of separation of concerns, SPEM2.0 keeps the usage of activities apart from their contents. It separates method content from its usage, where the usage is a reference towards the method content. Thus method content stays in the knowledge base and it is used in the process model through its reference. Contrary to SPEM's approach for realizing separation of concerns, we separate data-flow from control-flow in two distinct levels. We have taken inspiration from the object oriented paradigm and use the strength of typing concepts, so as to add both variability and conformance to our activity structure. An *ActivityDefinition* behaves like an activity type (more of a conformance than a type-instance relationship) and multiple *ActivityImplementations* can implement it. These activity implementations serve as a set of different alternative implementations for the activity definition. Each activity implementation carries its own properties. These properties are internal to an activity implementation and can not be accessed outside it, except through the specified contracts.

Contribution VII: *'Type-instance' like conformance relationship between activity implementations and activity definitions to foster variability in the process model.*

The restriction on interactions through the specified contracts only, adds the flexibility to choose any of the conforming alternative activity implementations for an activity definition. The choice of the right alternative may be deferred till execution (in the next phase) and the chosen alternative may be added on the fly. A new activity implementation must implement all the interfaces of the activity definition, thus avoiding any issues of compatibility. The modifications in an activity implementation do not affect other activities in the same context having dependence on it (as far as it conforms to the activity definition). Likewise, any modification to the context of an activity implementation does not effect its content. An activity definition defines the contracts of the activity at the type level. Conforming to this activity definition, multiple activity implementations can be developed. An alternative activity implementation can be developed for a team or an individual, or it may be outsourced, changing the activity implementation to a different perspective altogether. All these activity implementations remain valid and can be replaced with any other activity implementation that conforms to the activity definition. The use of multiple abstraction levels induces a process variability in the process models.

The contract in a process model is specialized to abstract and concrete contracts as shown in the core package in figure 4.5. Each activity definition specifies an abstract contract, whereas each activity implementation specifies a concrete contract. As there exists a conformance relationship, 'hidden' within the 'implements' relationship between activity implementation and activity definition, similarly there is a conformance relationship between the concrete contract and the abstract contract. This conformance relationship between concrete contract and abstract contract is pre-

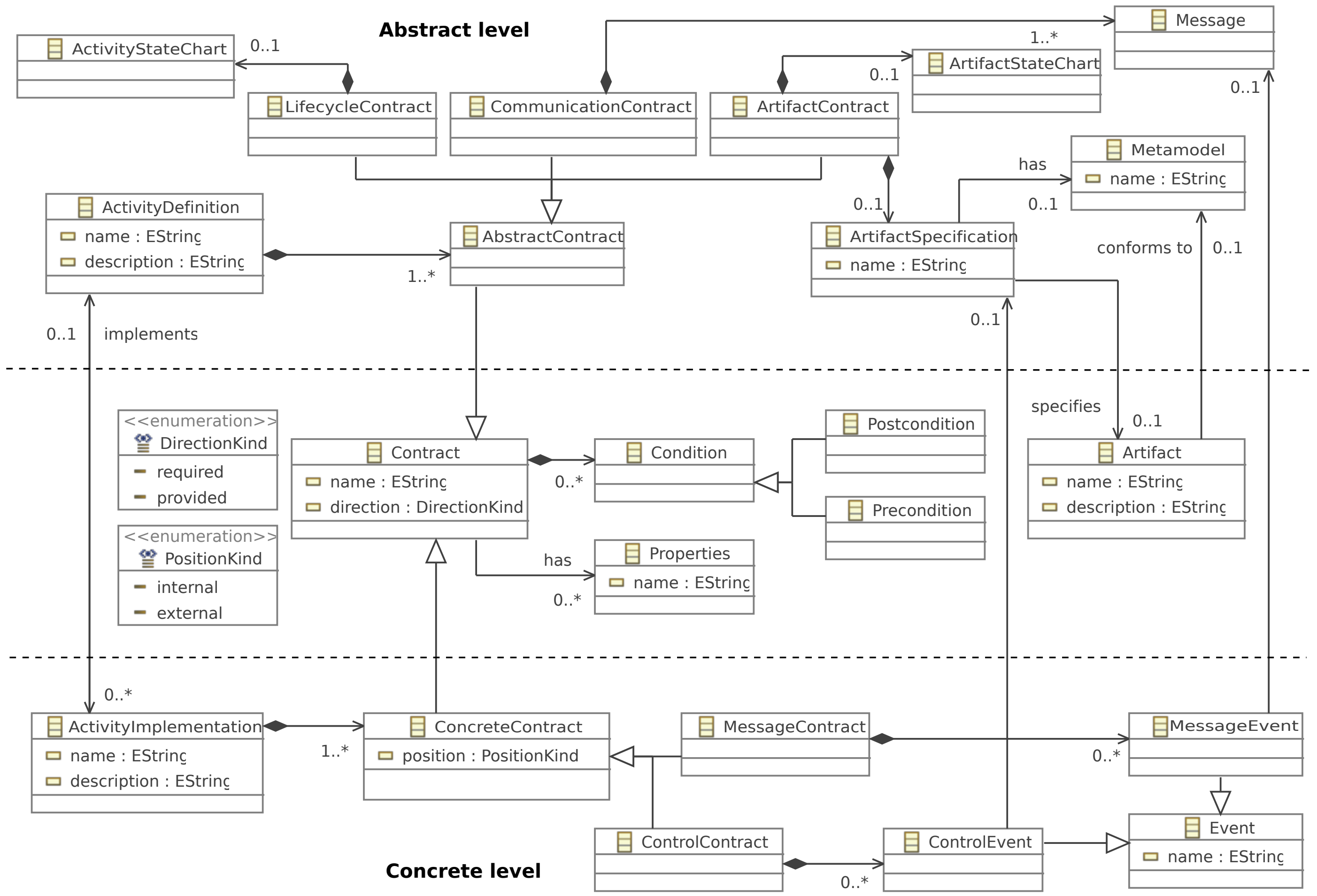


Figure 4.6 – Process Implementation Metamodel - Contract Package

sented through a mapping between the contents of the two entities, discussed in the *Contract Package* of the metamodel.

The responsibility assignment for a process is a very important aspect for process modeling. Responsibility, role and team entities are used to express these assignments for each activity. Each activity definition is played by a responsibility, whereas each activity implementation is played by a role. A responsibility of an activity definition specifies the level of authority for carrying out a task, for example approver or accountable. Whereas a role for the activity implementation specifies the precise function of a resource for the possible execution of the activity, in later phases of process lifecycle. A role enjoys the level of responsibility that it refers. Thus a project manager can be the approving authority for an activity and a software engineer may be responsible for performing it. A team is a composite role that has multiple roles as participants.

> **Contribution VIII:** *Allows to integrate different kinds of responsibility assignment matrices, supported by Project Management Institute (PMI) standards, in the process model.*

The second package of the PImp is the *contract package* which merges in the *core package* for the complete *Process Implementation Metamodel.* Contract package is presented in Figure 4.6. All the interactions to/from activity definitions or activity implementations are carried out through their specified contracts. These contracts can either be internal or external to the activity definition or activity implementation, which is defined through the position of the contract. The direction of the contract defines whether it is used for inward communication (required) or outward communication (provided). It should be clear that the interaction from sub-activities to a super activity is also inward for the super activity. In case of composite activities, for every external contract, there is a corresponding internal contract of the opposite direction. Thus for a provided external contract, there exists a required internal contract and vice versa.

An activity, being the basic unit of processing, takes work products as inputs, modifies or works on them and produces them as outputs. The input and output work products of an activity are called artifacts and they specify the data-flow within the process. An abstract contract of an activity definition deals mainly with the artifacts, whereas a concrete contract deals with the events. Abstract contract is specialized into three different contracts: *Artifact Contract*, *Communication Contract* and *Lifecycle Contract.* Artifact contract presents the *artifact specification*, which is used to describe the inputs and outputs of the activity at abstract level. Apart from specifying the artifact, it also presents the metamodel for the *artifact.* This separation of contractual resources, allows to separate the data-flow of the activities from their control-flow. Dealing with the data-flow at an abstract level (apart from the control-flow) allows the data-flow mechanisms to benefit from effective means like data repositories and configuration management. Whereas the control-flow within the activities through the use of events in concrete contracts can be effectively managed by an underlying event management system. One of the major reasons to choose event based mechanisms is to allow the decoupling of the activities for the control-flow. By decoupling these activities using events, the focus is brought back to the completeness of the definition of each individual activity. It also supports manipulations to the activities at runtime, as long as the contract is not broken. All the control events owned by the concrete contract map to the artifacts specified through *artifact specification* at the abstract level. Having two separate levels, a link is kept between the control-flow and the data-flow, so that the data-flow should be able to guide the control-flow among the activities, as and when necessary.

One of the key features of our framework concerning artifacts is the exploitation of structure. Artifacts are not taken as black box entities and their structure is kept comprehensible to the activity. This is enforced by the fact that each artifact conforms to a metamodel. This allows an activity to take an input artifact and even dynamically reconfigure itself (if needed) through the use of that artifact. The syntax and semantics encoded within the artifact is explicitly made accessible to the process that can exploit this information. This further allows us to define semi-automatic and automatic processes in our framework, for example model transformations. Artifact specifications also contain the *artifact statechart* that defines an automaton for the artifact states. Contracts at both levels define conditions: pre-conditions for required contracts and post-conditions for provided contracts. Using pre-conditions on the structured artifacts helps in managing semi-automatic processes with model manipulations. For example, a transformation activity takes a model as an input, and has the capability to verify its conformance to the respective metamodel. In case, the process modeling approach does not accept the artifact as a structured entity, an activity can not evaluate the input artifact based on its structure or the associated properties. The concept of using a structured artifact may not be of prime focus for manual activities, but is of high value for modeling the automatic and semi-automatic activities.

Contribution IX: *Use of structured artifacts in the process model so that activities can access the properties of the input artifacts.*

Lifecycle Contract is an abstract contract that defines the activity state chart. Activity statechart defines the activity lifecycle. Events defined at the concrete level of the process model conform to *ActivityStateChart* or *ArtifactStateChart.* Two kinds of concrete contracts for PImp model are *MessageContract* and *ControlContract. ControlContract* owns *ControlEvents*, which map to artifact specifications. Control events conform to *ArtifactStateChart. MessageContract* owns *MessageEvents* that map to the messages defined at abstract level. Message events conform to the *ActivityStateChart.* A detailed discussion about the contractual interaction is presented in section 4.3. *Communication Contract* is another absract contract that defines messages between the activity roles. A message event in the concrete level maps to these messages, and is responsible for the actual choreography between the activities/roles.

ISPW-Scenario in Implementation phase

ISPW process modeling scenario presented in section 4.1.2 is implemented using the PImp metamodel, as shown in figure 4.7. The specification model of the same scenario presents the activities only at the definition level. However, a process implementation model of this scenario presents both the abstract and the concrete level. The graphical notation for this model only presents the concrete level of the model with one implementation for each activity definition. However, an implementation phase process model can have multiple implementations for each activity definition. This model presents the *SoftwareChange* (SC) process, that composed only one activity, the *DevelopChangeAndTestUnit* (DCTU) activity. DCTU is a composite activity implementation, which composes a *DevelopChangeTest* (DCT)process. Both SC and DCT shown in the graphical notation are the concrete processes.

An interesting point to consider in this scenario is that the internal architecture of DCTU activity (which is composed of 8 sub-activities) was part of the (abstract) process in specification model. However, in the implementation model this becomes a single implementation of the DCTU activity. This allows to add other implementations of DCTU activity, which might compose less or more sub-activities or might even be a primitive activity. At the abstract level DCTU activity specifies its contracts and other details regarding *objectives*, *guidelines* and *responsibilities etc.* The abstract level process specifies the binding between the activities. For example the sub-activities of the DCT_Df process are connected together through directed edges between them. These directed edges depict a binding between the provided contract of the source activity and the required contract of the target activity. DCT_Df process with its sub-activities, connected together through bindings, presents a workflow at the abstract level. The implementation of the DCTU activity by the composition of DCT process is defined at the concrete level.

In the concrete level of the model, there is no binding between the contracts (the edges between the contracts in the graphical model are for the ease of understandability only). The required contracts express their dependency over the events related to the required artifact. Similarly the provided contract expresses its ability to trigger the events related to the provided artifact. Events are prepared to be broadcasted in the context or content of the activity implementation depending upon the position of the contract. For example, the internal provided contract *currentDesign* of the DCTU activity can broadcast any corresponding event to the content of the activity and the external provided contract of the *ReviewDesign* (RD) activity can broadcast the *designReviewFeedback* event in its context. The presence of events in the required contract of an activity specifies that it is listening for this event through this contract. For example, the *designReviewFeedback* contract of the RD activity is an external required contract. This contract specifies that RD activity is listening to the events corresponding to *designReviewFeedback*.

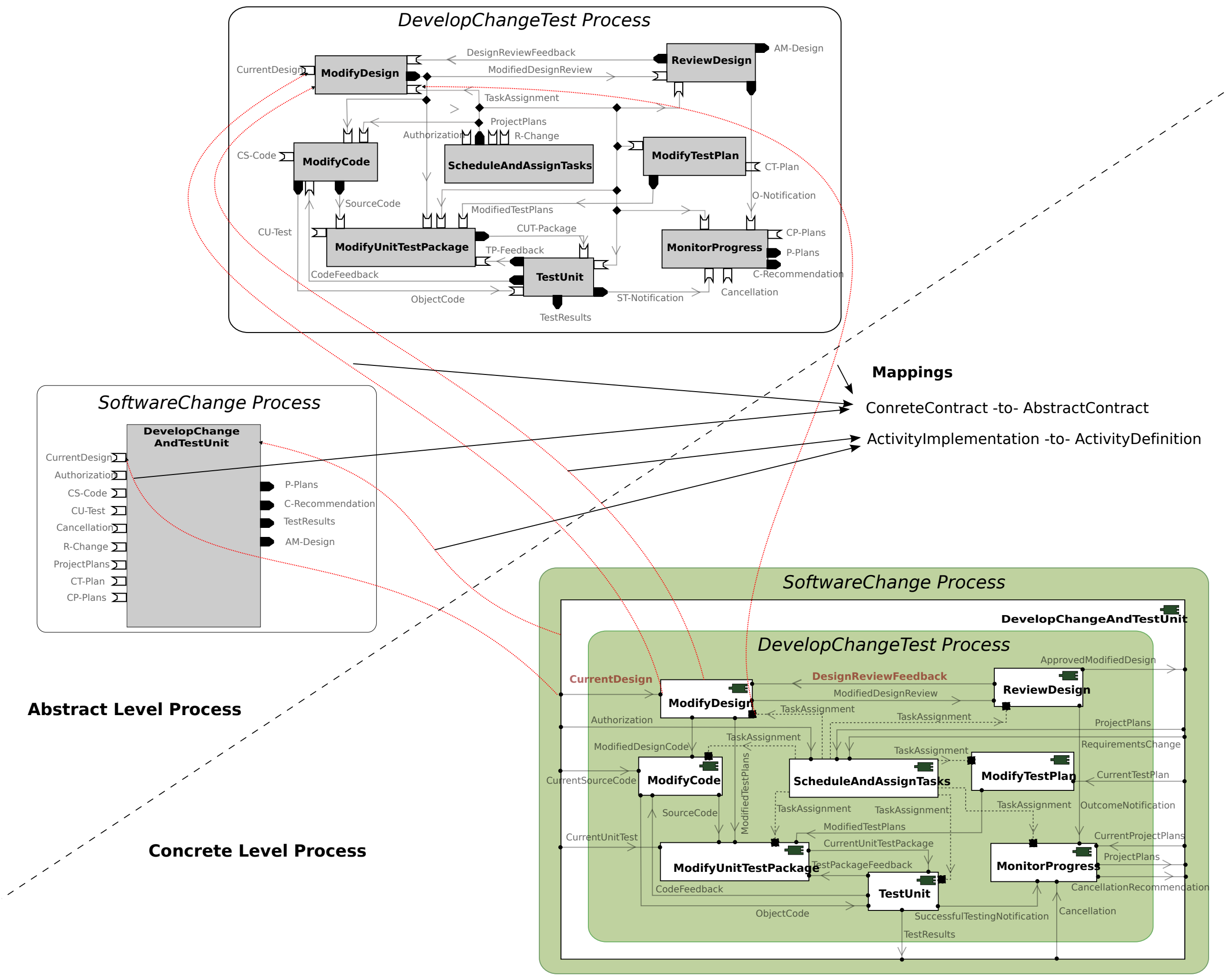


Figure 4.7 – ISPW Scenario - Implementation Phase

4.2.3 Instantiation phase

By the start of instantiation phase of process development lifecycle, processes are already built with multiple solutions. Multiple solutions are the alternative process implementations, where all of them conform to the activity definitions. Out of these multiple activity implementations, a subset of implementations is chosen to develop an executable process. These processes are not yet ready to be executed as they lack the link to real-life data. This linkage with real-life data, their scheduling and connections with the IT solutions give them the capability to be executed. Once they are ready to be executed, they can be enacted through the enactment/execution engines. The main objectives of process instantiation phase in software process development lifecyle are as follows:

— **Select the most feasible solutions** Multiple solutions for implementing each process may be developed in the implementation phase. These solutions are available in the process repository. Process engineers need to select the executing implementations based on different criteria like performance, contextual constraints, resource availability, *etc.* in this phase.

— **Link inputs/outputs to real data** The processes in this phase should be made executable and (where applicable) must be connected to the databases, document files, configuration management systems for work products, *etc.*

— **Review plan scheduling** Time duration of each activity is part of its implementation. But for developing an executable process model, activities need to be assembled according to the scheduling plan of the project. This plan needs to be developed at this phase, which affects the selection of appropriate process components.

— **Review cost scheduling** Apart from time scheduling, cost scheduling needs to be carried out at this level. Each process component implementation might require different level of resources and duration. Based on these properties, a cost analysis can be carried out for the project. This cost analysis is also one of the deciding factors for choosing appropriate implementations.

— **Linking processes to IT applications** Software development processes target the development of software systems. For carrying out these processing multiple tools are used. These tools can be invoked by the processes themselves. Linking the process components with their respective IT applications for the purpose of their execution is handled in this phase of process development.

— **Perform process testing** Each process component is already developed in the previous phase, where its unit testing is the responsibility of the previous phase. However, the integration of these process components into a complete executable process is carried out in this phase. Integration testing of the processes is also carried out in this phase so as to verify the correctness of the processes.

— **Roll out processes** Once the process components are integrated to develop an executable process, they are made available for execution. Process roll out is the release of executable process components which may be reused from the process repository.

Process Instantiation Metamodel

Processes are already designed and implemented before the Instantiation phase. *Process Instantiation Metamodel* (PIns) focuses on the execution semantics of the process model. As with the previous metamodels, its structure also revolves around the concept of abstraction. This abstraction allows the activities to depend on the contractual specifications of other activities, rather than on some specific concrete instances. This abstraction is expressed in terms of a bi-layered structure, where the upper layer is the abstract level and the lower layer is the concrete level. This abstraction boundary is crossed by a type-instance-like relationship (implementation) between instantiation activities and activity definitions. Other relationships initiating from concrete level that cross this boundary also pertain to conformance or realization of some abstract notions. For example, the *materializes* relationship is responsible for the realization of artifact, based on the specifications described in artifact specifications, at abstract level.

PIns is also developed using two packages, where the Instance-contract package merges with the Instance-core package for the complete metamodel. A process in *Instance-core package* is defined as an assembly of activity definitions or activity instantiations depending upon its level of abstraction, as shown in figure 4.8. This figure highlights the concepts that are different (updated or added) in PIns metamodel from the PImp metamodel. Hierarchies are managed in PIns metamodel in the same way as in PImp metamodel, using primitive and composite activities. An abstract process contains the assembly of activity definitions, where each activity definition behaves as a *type* for a set of instantiation activities. *Activity Definitions* do not compose any other process or activity definitions directly. The are implemented by the instantiation activities, which decide whether this activity is composite or primitive. Thus an activity definition can be implemented by either a *primitive instantiation activity* or a *composite instantiation activity*. A composite instantiation activity in turn contains the *instance process*. An instance process in turn contains both instantiation activities and the activity definitions. A very important structural difference between the PImp and PIns is the content of a concrete level composite activity. A composite activity implementation in PImp only contains concrete process with activity implementations mapping to activity definitions. It does not contain the activity definitions. On the other hand, a composite instantiation activity from PIns contains both the abstract and the concrete process, which means that it contains a complete process model in itself, where no instantiation activity maps to an activity definition out of its context. It allows an instantiation activity to be complete for execution.

Similar to PImp metamodel, activity sharing is support in PIns metamodel as well. The concrete level of the instance process model defines the concrete processes

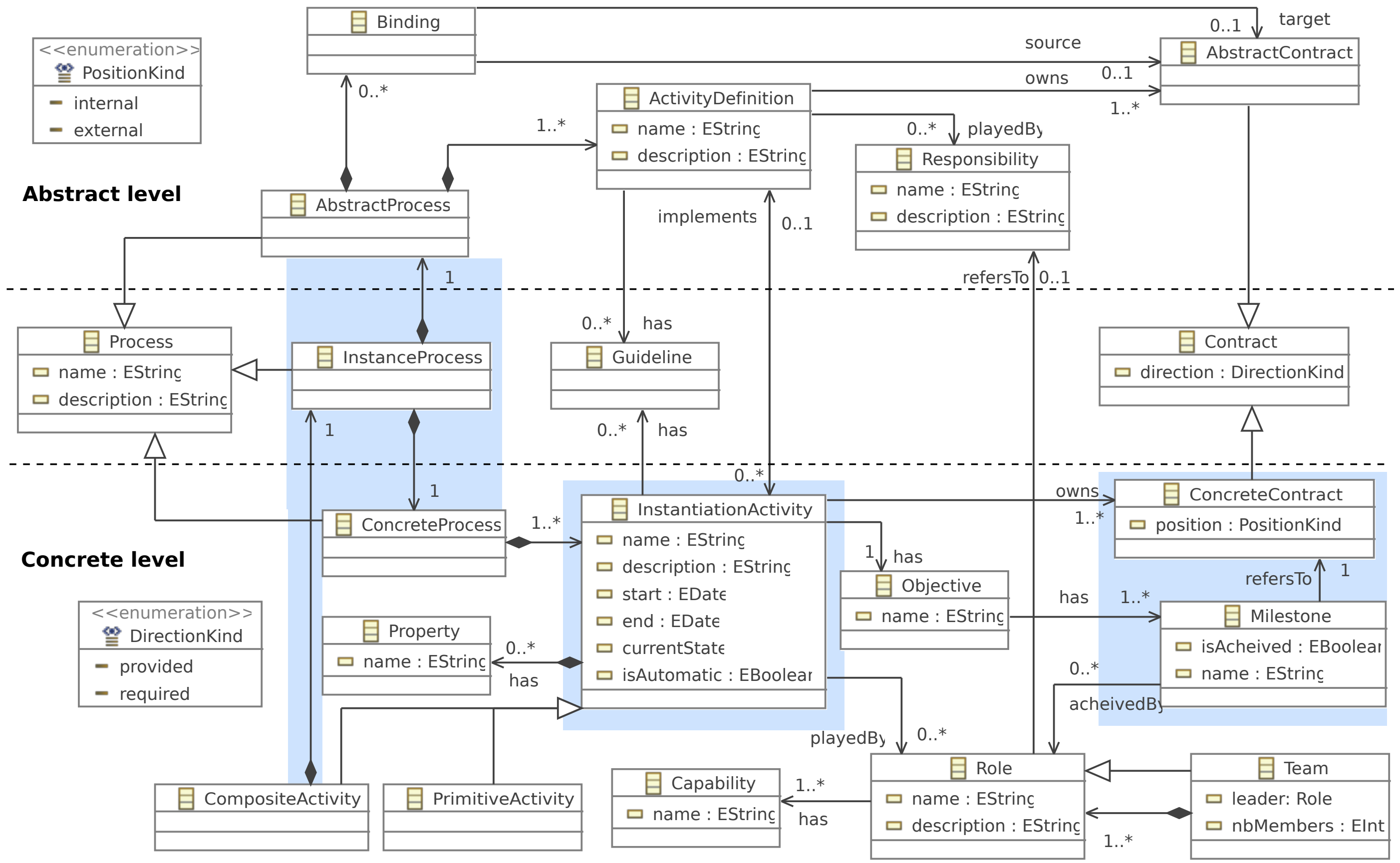


Figure 4.8 – Process Instantiation Metamodel - Instance-core Package

within a model. These concrete processes follow the hierarchical structure of composite and primitive activities. Besides implementation details, instantiation activities also define the instantiation details of the activity like duration, start date, current execution status, iterations and current iteration etc. An instantiation activity defines the concrete contracts for interactions with other activities. These concrete contracts like abstract contracts can either be required or provided. Activity definitions do not compose any other process, so all the contracts are external. However, instantiation activities may implement an activity definition as primitive or composite activity, so their contracts may be positioned internally or externally. Internal contracts are only present in the composite instantiation activities. For every contract of a composite instantiation activity, there exists another contract with opposite direction and opposite position. This allows for the delegation of contracts from external context to the internal content or vice versa.

A notion of responsibility is specified for performing each activity definition at the abstract level. It specifies various kinds of responsibilities that need to be taken by the roles *i.e.* responsible, accountable, signatory, *etc.* The responsibilities are assigned to the *roles* that play the instantiation activities. Each role is a collection of capabilities needed to carry out the activities. These roles can be composed to form teams. Each *role* is enacted by an actor or tool. An actor is an human resource (*i.e.* employee/working for the organization) that performs the manual activities or the manual part of the activities. The automatic part or the automatic activities are performed by tools. These tools are the IT applications which are used to perform various tasks related to the software development processes. Roles are linked to the provided contracts through *milestones*. A milestone links the objective of an activity and its role to the concrete level provided contract. Milestones are used to monitor the progress of the executing process and the work performed by the actors.

Instance-contract package of PIns defines the contracts of activities both at abstract and concrete level, as shown in figure 4.9. Similar to PImp metamodel, abstract contracts are specialized as *Artifact contract*, *Lifecycle contract* and *Communication contract*. Each *artifact specification* specifies a metamodel for the artifact, which means that each artifact is considered as a model. Each artifact conforms to its respective metamodel. Thus an activity has access to different components and properties of an artifact, because each artifact is structured. The concept of *metamodel* is depicted through a single class in PIns for reasons of brevity. For the development of an instance process model, it is the responsibility of the process designer to provide the associated metamodels for each artifact. Artifact contracts also contain the *ArtifactStateChart*, which gives the automaton for the different states of an artifact. These artifacts are kept in the artifact repository, so each artifact specifies its repository address. An artifact can either be a hard copy or a digital document. Each digital artifact in PIns has a unique repository URL, which is used for access it in the artifact repository. Artifact repository supports versioning, thus an artifact can be locked by an activity, depending upon the nature of the activity.

Contribution X: *Allows the flexibility to define a custom lifecycle for an artifact.*

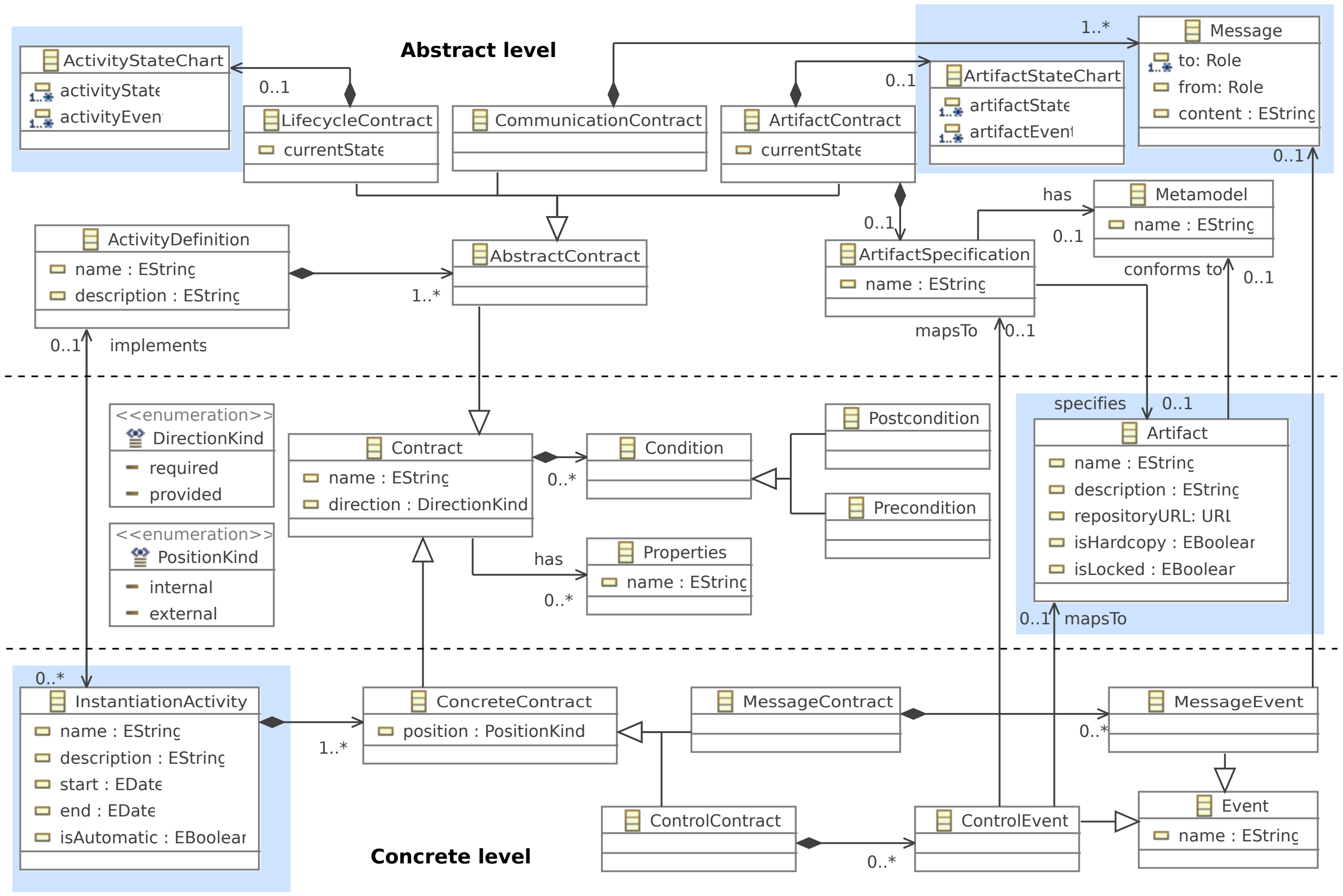


Figure 4.9 – Process Instantiation Metamodel - Instance-contract Package

Abstract level of the process model defines two State machines, one for activities through *Lifecycle contract* and the other for the artifacts through *Artifact contract*. These state machines are public to other processes in the context of implementation phase of process development lifecycle. Their validation is carried out at the implementation phase. In process instantiation metamodel, these state machines are kept private and only a list of events is made public. *Communication contract* defines the messages that can be sent between the roles of different activities. These messages are defined in the abstract level, however they are triggered from the concrete level through message events. This way, messages can be predefined and sent when they are required.

Mapping of control events in the concrete level to the artifacts aligns the control flow of the process with its data-flow. The event management system implemented in the process interpreter handles the execution of processes at runtime. Using event driven execution for the processes allows reactivity for the process model at one hand and dynamic process reconfigurations at the other. Specification of the data-flow and control flow in two separate levels and then mapping them together results in a process framework which is proactive for data-flow, yet reactive for the control flow. Though artifact specifications are defined at the abstract level, the concrete artifacts that materialize those specifications are defined at the concrete level. Thus the data-flow remains at abstract level but the exchange of real life artifacts between the activities is handled using events at concrete level.

The motivation of using such a bi-layered structure for the process framework is manifold. First, it allows to deal with the process variations where addition/manipulation/ improvement of activity implementations is possible, even at runtime. Second, it supports a correct by construction approach for the executable activities. Third, it helps in separating the concerns in respect of data and control for an executable activity. Fourth, it supports re-usability of process at varying levels of abstraction *e.g.* we can reuse either an activity definition or an activity implementation. Finally the use of abstraction allows better means to organize and synthesize existing processes in a formal process model.

ISPW-Scenario in Instantiation phase

We have presented the ISPW process modeling scenario of section 4.1.2 in continuity from specification model to implementation model. This section presents it in the instantiation phase, as shown in the figure 4.10. Composite activity Implementations from the PImp metamodel, contain the concrete process, whereas the instantiation activities of the PIns metamodel contain both the abstract process and the concrete process. This results in a self sufficient, complete activity that is ready for execution. The graphical notation for this model takes one instantiation activity for each activity definition, however other alternative instantiation activities are also present in the model. These activities can be replaced with each other through a proper mechanism of runtime adaptation, that will be discussed in the next chapter. *DevelopChangeTestProcess* is an abstract process contained within the *Devel-*

opChangeAndTestUnit (DCTU) activity and the corresponding concrete process is also contained in the same activity. The activity definition for DCTU activity at the abstract level is the only activity definition present in the *SoftwareChange* process.

The concrete level and the abstract level activities are both present in the parent instantiation activity. DCTU activity contains the abstract level activity definitions for all the activities that it contains. The abstract level of the process remains the same as in the PImp model. There is a mapping between each entity of the PIns model to each entity of the PImp model. This is used to trace back the origins of each entity, till the specification model. This is crucial when a process model is refined over multiple phases and one wants to know which activity is following which specific specification model. In this example we took one specification model and refined it till the instantiation model, but it is possible to take two specification models and develop a single implementation model from the two. In such situations, tracing back to the original entity in a particular specification model is important.

The concrete level shown in this graphical notation does not link the contracts together. The reason for this notation is to show that there is no explicit specification of the dependency between the activities. An instantiation activity expresses its dependency on specific events that might map to a particular artifact. For example the *ReviewDesign* (RD) activity is expecting to receive a *modifiedDesign* document for review. RD activity is not depending on a specific instantiation activity that provides this document. Any activity that produces this *modifiedDesign* document can provide this document. This creates a decoupling between the instantiation activities, where the focus remains on the complete definition of each activity in a process model.

Events within the DCTU activity are managed by the event broker provided by it. This event broker takes care of the routing of events between the instantiation activities in a manner that AND, OR and XOR connectors can be simulated. *ModifyDesign* (MD) activity provides the design document to three instantiation activities: *ReviewDesign* (RD), *ModifyCode* (MC) and *ModifyUnitTestPackage*(MUTP). MC and MUTP activities need this design document only when it has been approved by the RD activity. Even if the document availability notifying event is received, the pre-conditions of the MC and MUTP activity do not let the document pass through the required contract. It is only possible to access this document when it is approved. The input contracts of the RD activity require *currentDesign* and *DesignReviewFeedback*. There is an OR-type logic specified within these required contracts, such that the activity can start its processing from any of these events (when the availability of *currentDesign* is notified by artifact event).

A mapping is shown in the graphical notation for only one sub-activity, *TestUnit*. These mappings between the concrete level process and the abstract level process are implicit in the CPMF model. So they are not shown in the model explicitly. In the figure 4.10, this mapping is shown for discussion purpose only. The 'circle-shaped' contracts in the concrete level of this model represent the provided contracts of the activity, whereas the 'square-shaped' contracts represent the required contracts. There is a mapping between the concrete contracts of the *TestUnit* instantiation activity and its associated activity definition. Control contracts are specified using

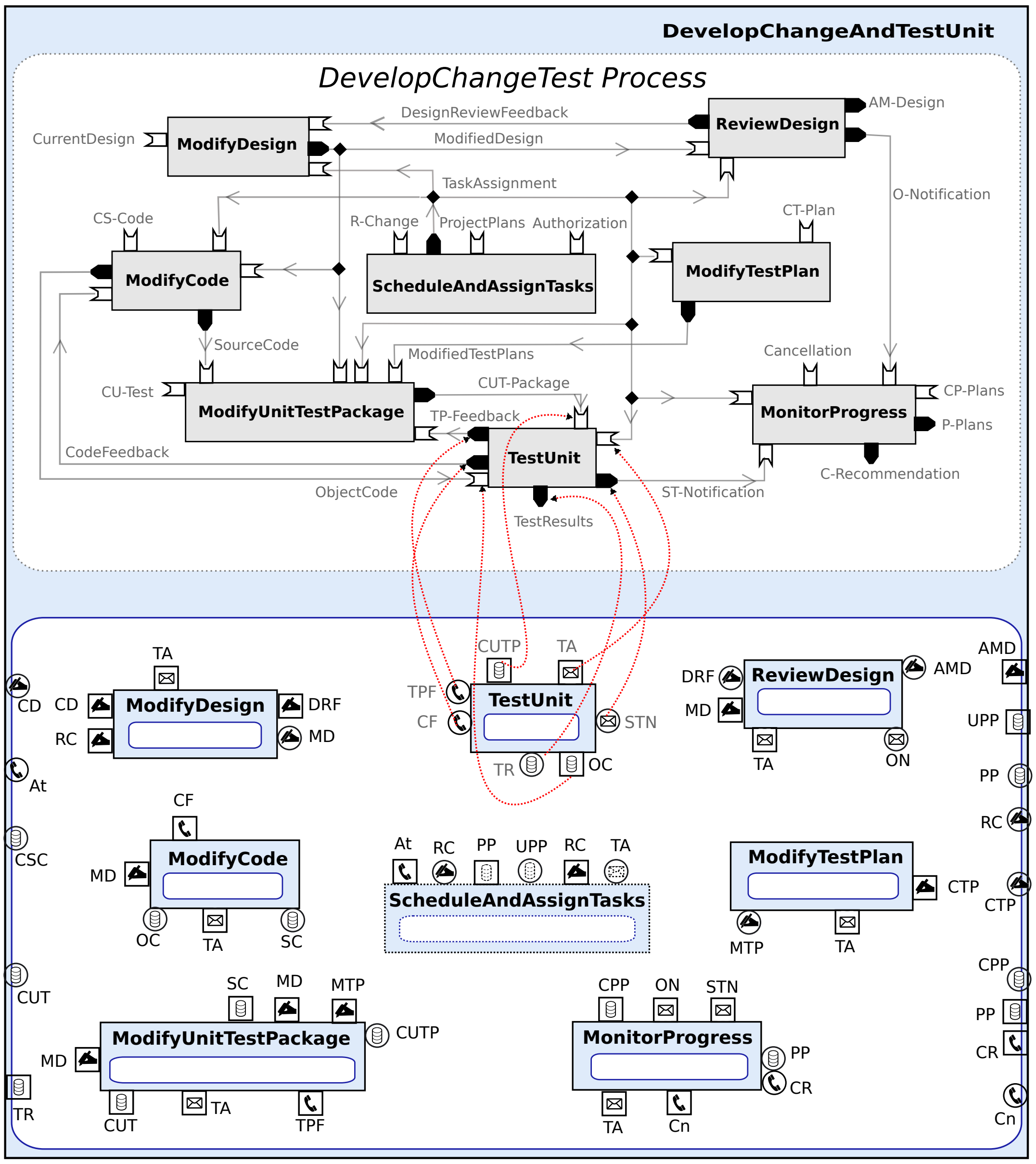


Figure 4.10 – ISPW Scenario - Instantiation Phase

'hand' symbol for hard-copy hand carried artifacts and 'data store' symbol is used for the digital artifacts stored in the repository. Message events are also either by emails or by telephone. But these symbols are not of prime importance and depend upon the tooling support implementation. The important point to note is that the model is refined to the level that each interaction is properly defined.

4.3 Contractual Interactions

4.3.1 Design by Contract

Design by Contract (DbC) is a software construction approach that was developed in the context of software programming language Eiffel [Meyer 92b]. The prime focus of this approach is to guarantee the valid interactions between software components, through the use of contracts [Meyer 92a]. Software components are taken as clients (callers) and suppliers (routines). A client demands a service and a supplier provides it. The mutual obligations and benefits between clients and suppliers are explicitly specified. A supplier is obliged to provide "valid response", if all the conditions for its processing are duly met. Similarly a supplier has the benefit to receive only the "valid requests" which bind it to respond with the valid response. In case of software process modeling, an activity plays both the roles; clients and suppliers. When an activity receives an input from some other activity, it serves as a client of that activity for that specific input artifact. When an activity provides an output to some other activity, it serves as a supplier of that activity for that specific output artifact.

DbC uses the concept of *Abstract Data Types* (ADT) to ensure the encapsulation of the data and methods behind well-defined interfaces. Assertions are exploited to explicitly specifiy the interfaces for the components through pre-conditions, post-conditions and invariants. Specification of these interfaces is called a 'contract'. These contracts defined in ADT are bound together for the interacting components. Activities defined in the CPMF framework follow the DbC concept and specify their interfaces for interaction with other activities. This ensures the encapsulation of data and 'methods' of a software activity. When all the interaction of an activity is restricted through the specified interfaces, the use of conditions guarantee the validity of inputs and outputs for it. For example, if a software development activity is responsible to review a piece of code for quality assurance, it can specify the input software development file and the output software development file through respective interfaces. The input interface of the software development file can add pre-conditions *e.g.* the code should be executable, it should conform to a specific language grammar, etc. Similarly the output interface can add the post-conditions *e.g.* the software development file should be approved by quality control engineer, it can even specify the quality metrics for redundancy, cycles, nested block depth *etc.*

The core idea of design by contract is to hold responsible the software components that is developed to carry out a particular task. This means that a software component is responsible to provide the expected results if it is provided with the inputs that it expects. This helps in locating the precise component in case the output of a system is not what was being expected. Anomalies in the software processes can be located and fixed precisely and efficiently by checking out which specific contract between the activities was broken. It also promotes the completeness of each individual activity by explicitly specifying the inputs and the expected properties of inputs through pre-conditions. Similarly the outputs and the desired properties of outputs are ensured through the post-conditions. Focus on the completeness of definition for the activities promotes explicit coupling. This explicit coupling is not a dependence of an activity on some other activity, rather on some well defined inputs.

One other important motivation for choosing DbC for the CPMF framework is to promote reusability. The overall architecture results in a modular process, where each activity serves as a separate module. Each module in this architecture has its defined interfaces, coupled with well specified assertions. This way, any activity that specifies compatible required and provided contracts can be replaced with the current activity. Compatible contracts mean that the new activity can have less required contracts but it should offer all or more provided contracts than that of the activity it is replacing. The activity being replaced and the replacing activity should both have the same objective as well. It means that once an activity is developed in CPMF, it can be used in any location, where it has access to the required contracts, provides the desired contracts and serves the objective. This fosters reusability in process modeling approach, provided a means of storing, searching and retrieving existing activities is offered by the implementing tools.

Apart from modularity and reusability, DbC can also be exploited to offer multiple abstraction levels. These abstraction levels serve as the concept of interfaces in software programming languages, where the specification of interfaces defines a type for the software module. Any concrete level software module that implements these interfaces has to offer the implementation of these interfaces. This way, all the implementations of the same 'type' are guaranteed to provide a known set of outputs. This concept of abstraction has been exploited by CPMF in its implementation and instantiation level metamodels. Besides the level of abstraction, the structure of a contract also changes with advancing phases of process development lifecycle. This is visible from the refinement/evolution of contracts over the process lifecycle.

4.3.2 Contract refinement

CPMF follows DbC to have a modular process architecture with specified interfaces. Each activity defines its contract to be able to interact with its context or content. *Context* of an activity are the sibling activities and the parent activity whereas *content* of the activity are the activities composed by the process that it contains. The contracts of an activity have a *direction* and a *position*. The direction of a contract specifies whether it is required or provided and the position of a contract specifies whether it is internal or external. Internal contracts are only possible for composite activities. For composite activities their contracts are in the form of a pair. For each external contract, their exists an internal contract of opposite direction. This pair of contracts for composite activities serves for delegation purposes. It is used to pass the interaction from the content of an activity to its context or vice versa.

The contracts of the activities in CPMF are responsible for a guaranteed interaction between the activities. Each contract of an activity adds an 'obligation' or a 'benefit' to the activity. *Obligation* of an activity is its provided contract, which obliges the activity to provide the valid *contract content* specified by the contract, if all its requirements for processing are met. *Benefit* of an activity is its required contract, which guarantees to pass through only the valid *contract content* to the activity. *Contract content* of a contract is specified by it and depends upon the level of abstraction of the activity. This mutual agreement on the obligation of one activity towards the benefit of another activity creates a dependency between them. The guarantee of correctness of interaction for every dependency comes from the notion of this mutual agreement. This dependency between the activities may or may not be explicitly specified in the model, depending upon the level of abstraction. Dependency between abstract level activities is explicitly specified. However, on concrete level the dependency between the contracts is not specified. The reason for not specifying the dependency is that an activity at concrete level can receive the input from any other activity that provides 'correct' output.

The core idea of CPMF framework is to have a refinement-based process modeling approach, where a process is specified in its initial phase of development and is refined over time with advancing phasing of process development lifecycle. For the current implementation of this framework, we chose three metamodels, each for a different phase of process development. Each of these metamodels evolves the structure of the process. The contracts of activities in these processes are also refined along the development phase.

Initially, the processes are specified using the *Process Specification metamodel* (PSpec). PSpec defines the activities having two types of contracts: *Communication Contracts* and *Artifact Contracts*, as shown in figure 4.11. Communication contracts of specification activities define the messages that need to be passed between activities for carrying out the processing. Artifact contracts of a specification activity contain the *artifact specification*. Artifact specifications specify the artifact that is required or provided by the activity.

Activities at the implementation level of process development are of two kinds: *Activity Definitions* and *Activity Implementations*. *Activity Implementations* provide concrete implementation methodology either by composing other activities or by defining the primitive level tasks to accomplish it. The contracts of activity implementation are the *concrete contracts* and the contracts of the activity definition are the *abstract contracts*. When the contracts of a specification model are refined to the implementation model, they transform to abstract contracts. Concrete contracts in process implementation model are developed in the implementation phase from scratch. Concrete contracts need to conform to the abstract contracts, so that the activity implementation can conform to the activity definition.

Abstract contract in PImp are of three kinds: *Communication Contract*, *Artifact Contract* and *Lifecycle Contract*, as shown in the contract for activity definition in

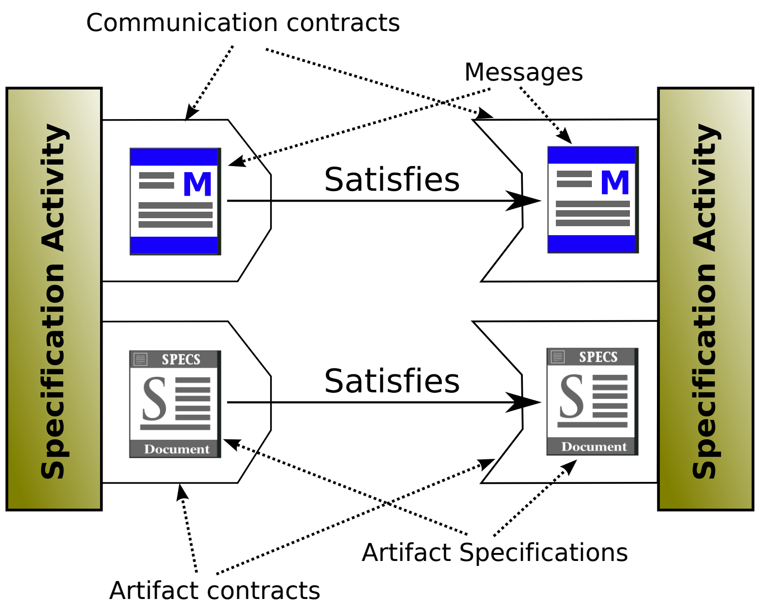


Figure 4.11 – Contracts for Specification Activities

figure 4.12. A communication contract in a process implementation model is a direct mapping from specification model with no change in its structure. It defines the message based interaction between two activities. For example, if an activity requires an information from some other activity to continue its processing, it is specified through a communication contract. Different standards, use messages to define the interaction between the activities. These communication channels are specified through these contracts. An artifact contract of a process implementation model results from a transformation mapping from the process specification model. An artifact contract is refined in implementation model by adding an artifact state machine, defining the automaton of the artifact lifecycle. The same artifact can have different lifecycles for two different activities *i.e.* the artifact provided by an activity can follow a lifecycle within that activity and another different in some other activity that requires it. For example a software development file with some code may be completed for an activity that produces it. When this file is required by quality assurance activity, it might follow a different lifecycle till it is approved by this activity. For a binding between a required artifact contract and a provided artifact contract, their must be a 'satisfaction' relationship between the artifact specifications in them and a 'subset' relationship between the provided contract and the required contract. 'Satisfaction' relationship guarantees that the provided artifact serves the need of the activity that requires it for carrying out the processing. The 'subset' relationship between the two artifact state machines guarantees that the events specified by the provided contract can all be understood by the state machine of the required contract.

Lifcycle contract in PImp is also an abstract contract that defines an activity state machine. Activity state machine defines the automaton for the activity lifecycle. It defines the events and the respective states that an activity can attain. This contract is used to trigger the processing of its activity and to ensure the synchronization with

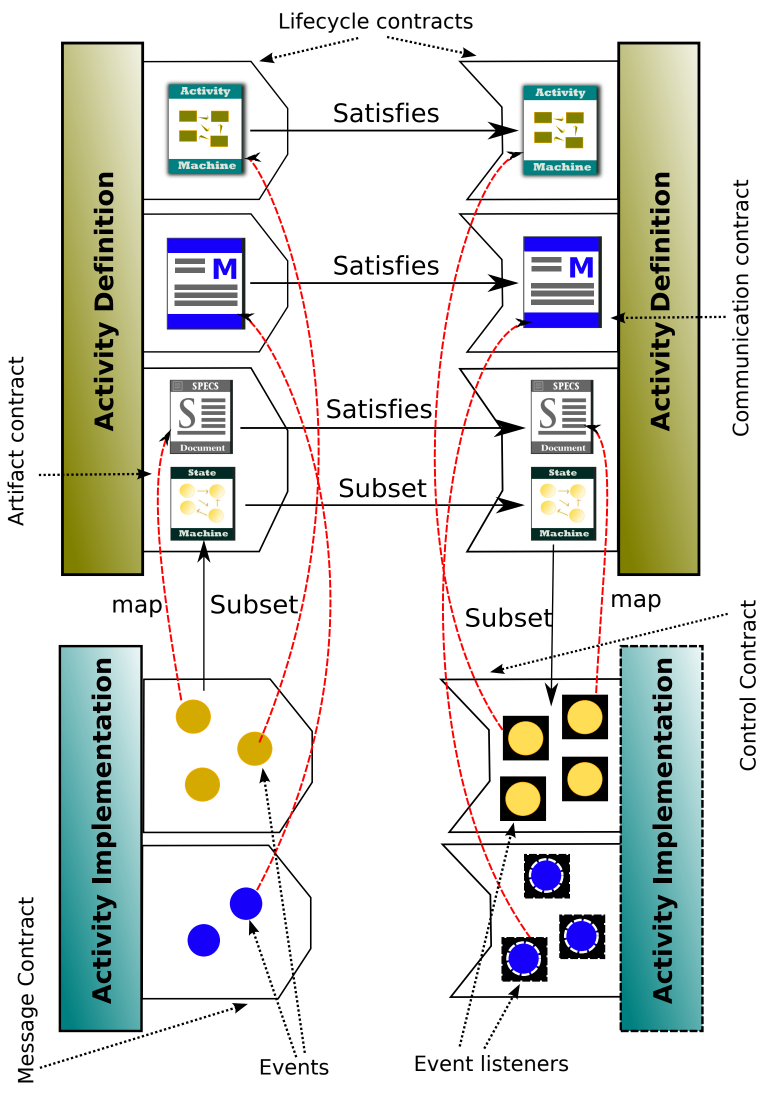


Figure 4.12 – Contracts for Implementation model Activities

the parent activities. For example, the termination of a maintenance activity would terminate all its sub-activities, with the exception of those that are being shared by some other parent activity, still in execution.

Concrete contracts in PImp define events. These events are responsible for the actual processing of the activities when the process is executed in the next phases. There are two kinds of concrete contracts defined by PImp: *Message Contract* and *Control Contract*. Message contracts define *message events*. These events are mapped to the messages defined in the communication contract at the abstract level. They are responsible for triggering the flow of messages between the interacting activities. Control contracts define *control events*, which may be mapped to the artifact specifications at the abstract level. When an event is mapped to an artifact specification, it is responsible for a change of state of the artifact (when the process is enacted in future phases). A control event that does not map to an artifact specification is responsible for the change of the state of the activity. Event in the required contract of an activity implementation is basically a listener to that event. Thus provided contracts are responsible for triggering those events and required contracts are responsible for listening to those events.

Events are specified in the state machines of the abstract contracts, however their definition is carried out at the concrete level. A provided *control contract* of an activity implementation defines a set of events that form a subset of the events specified in the artifact state machine at abstract level. This ensures that all the events defined in the provided control contract are already specified. As explained earlier, the events specified in the provided artifact state machine are a subset of the events specified in required artifact state machine. This ensure that if two activities are bound together for an interaction at the abstract level, then the activity receiving the artifact can listen to all the events specified for that artifact. The required control contract of an activity implementation defines the listeners for the events related to the artifact. The set of events specified by the required artifact state machine should be a subset of events for which the required control contract defines the event listeners. This ensures that the required control contract of an activity can listen to all the events produced by the provided control contract of the activity on which it depends.

An activity defined in the *Process Instantiation Metamodel* (PIns) follows the same bi-layered architecture of PImp *i.e.* it also has an abstract level process containing the *activity definitions* and a concrete level process containing *instantiation activities*, as shown in figure 4.13. Activity definitions are at the abstract level and have abstract contracts for interactions. Instantiation activities have concrete contracts, that conform to the abstract contracts of the corresponding activity definition. Abstract contracts of PIns result from a transformation mapping from the abstract contracts of PImp. Communication contracts at instance level are the same as implementation level, they both define the message that is used for the process choreography. Additional properties like *timestamp*, *medium*, *etc.* are added to the message but the structure remains the same. *Artifact contract* in PIns carries forward the same artifact specification. However, artifact state machine is only offered in the provided artifact contract. The required artifact contract only shows the list of events that it is listening to. The artifact state machine is still present in the activity definition but is not made public through the contracts. This promotes information hiding, when it is not relevant in instantiation phase, as all the dependencies are already specified at the implementation phase.

The provided *Lifecycle contract* of the activity definition in PIns offers the activity state-machine, just like in PImp. However the required Lifecycle contract does not offer the activity state-machine, instead it offers the list of events that it is listening to (similar to the required artifact contracts). The implementations of an activity are developed and validated in the implementation phase of process development lifecycle. Process Instantiation phase looks forward to add the execution semantics of the existing process implementations. It hides the details not required for the execution phase. The validation of contracts is also not repeated in the instantiation phase. For example a provided contract of a *prepare design* activity offers an artifact specification for a *design document*. A required contract from the *design review* activity also defines the artifact specification for this design document. A binding between the two contracts explicitly expresses the dependency of the *design review* activity over the *prepare design* activity. The 'satisfies' relationship between the two contracts enforces a validation check, which is based on two things: 1) the artifact in the two contracts points to the same 'real life artifact' and 2) the verification that the set of events specified in the artifact state-machine of the provided contract of *prepare design* activity is a subset of the set of events defined by the artifact state-machine of the required contract of *design review* activity. This validation between the binding contracts is not carried out again in the instantiation phase to eliminate redundancy.

The concrete contracts of the Process Instantiation model are the same as in the process Implementation model. The *Message contract* defines the message events and the *contract contract* defines other control events for the processing of the activity. However, the validation of conformance is already carried out in the implementation level, so the set of events defined in the concrete events are not validated against the abstract contracts anymore. The control events map to the artifact specifications in the implementation level model. In the instantiation phase, artifacts are prepared and stored in the artifact repository. For the 'hardcopy' artifacts, a reference is still kept in the artifact repository. The provided control contract of the instantiation activity maps directly to the artifact in the repository instead of its artifact specification. However, the required control contract of the instantiation activity still maps to the artifact specification of the artifact. An artifact in the artifact repository maps to the artifact specification that it realizes. Thus the transfer of artifact between two activities is handled through the control events, which notify the 'completion' of the artifact by the providing activity, after releasing its locks (if present) through the version management system. It is a choice of the activity implementer to decide whether or not to lock the artifacts.

The external control contract of an instantiation activity is responsible for triggering the events to its context, or for listening to the events received from its context. Internal control contracts are only possible for the composite instantiation activities. The internal contract of an instantiation activity also contains an event broker to

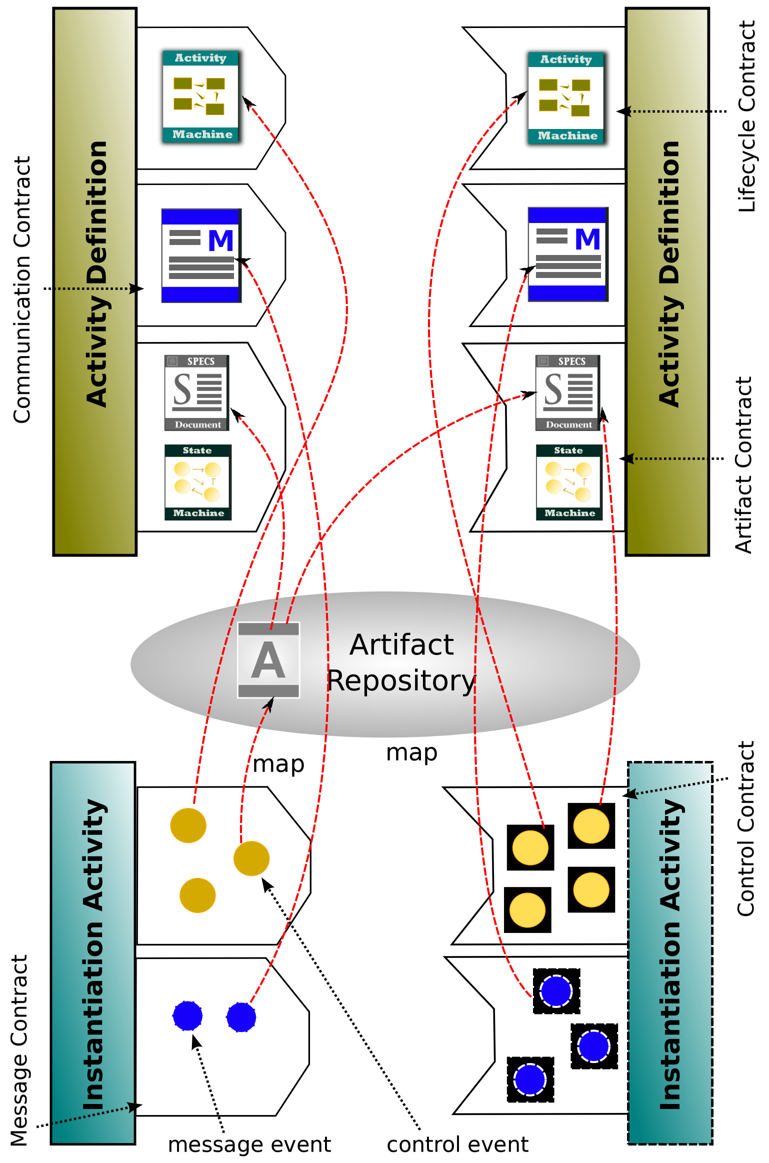


Figure 4.13 – Contracts for Instantiation model Activities

manage the events of the process that it contains. Events are broadcasted in the context of the contract that triggers them (unicast and multicast are also possible in the associated implementation). No logical connectors are defined in the abstract level of the CPMF process models. All the required abstract contracts follow an OR-type logic and all the provided abstract contract follow an AND-type logic. On the concrete level, any of the logical connectors OR, AND and XOR can be implemented by specifying the desired behavior in the respective contract. The event broker of the process takes care of the proper routing by keeping track of the successful interactions between the activities.

4.4 Methodological Summary

CPMF approach presents a methodology where multiple metamodels can be used for modeling the processes. Each of these metamodels targets a specific phase of process development lifecycle. A refinement relationship between these metamodels allows to refine the process model with each passing phase. The approach itself is not restricted to any specified number of phases. However, metamodels are provided for a basic process development lifecycle with only three phases of development: specification, implementation and instantiation. This approach can be applied to a more complex process development lifecycle where additional metamodels can be developed.

Process model at specification phase of process development takes into account all the constructs needed to specify its data-flow and interactions between multiple roles of constituent activities. This model is refined in the implementation phase where the control-flow of the activities is added to the process model. Multiple specification models can be created for single process model. These process models can be refined as a single implementation process model. The use of a bi-layered architecture in implementation process model allows to separate the data-flow from the control-flow of the process. Activities of the specification model are refined to activity definitions and activity implementations are defined for each of them. A default activity implementation of each composite activity definition presents the composition of its internal process (as specified in the specification model). Other activity implementations can also be developed for an activity definition. Other implementation level details include activity state machines, artifact state machines and objective *etc.* This model is then further refined in the instantiation phase to develop a process instantiation model. Process instantiation model adds the instantiation level details like scheduling information and links the activities to real artifacts, actors and tools. Activity implementations are refined to instantiation activities. Instantiation activity, as opposed to activity implementation, contains both abstract and concrete processes making it a complete unit of processing. Each activity is mapped to the corresponding activity of the previous phase of process development. This allows an instantiation activity to trace back to corresponding process specification.

This process modeling methodology can be applied in the software industry in different manners. First, following the methodology from scratch to develop the processes. Second, translating existing processes to CPMF process model and taking advantage of the different offerings of the approach. In the first case, a process designer is responsible to develop a process specification model. This model is an abstract model which can be developed from organizational process standards or any other adopted standard. In case, no standard is followed, a specification model can be developed from scratch for the precise project as well. The intention of developing process specification model is to have organizational level (or even a wider level) reusability of process specifications. Business analysts and other business professionals use this model to analyze the strategies of the business. In software industry, the business analysts can define organizational standards and strategies for the development of software.

Process implementation models are used for specific projects. The process designer is responsible for refining process specification models to develop a process implementation model. This project specific model contains the implementation details related to that project *e.g.* availability of resources and primitive level activity implementations according to the project requirements. Process designer can also choose to translate a process model from any other approach like SPEM, BPMN or YAWL to process implementation model. Currently, we do not offer any transformations to/from other approaches and thus the process designer will have to write such transformation definitions himself/herself[1]. In this case, the process designer can benefit from the existing process specification model to ensure compliance to organizational/adopted standards. Instantiation details can be added to this process model, once process designer refines the model for instantiation phase. Process instantiation model is executable and can be loaded into the process interpreter. Once the process is executed, then the stakeholders (organizational heads, project managers, business analysts, actors) can interact with it through the project management dashboard.

1. This transformation should be straight forward, keeping in view the table of corresponding constructs support in Appendix B

Chapter 5

Implementation of the Framework

Contents

Abstract - *This chapter presents the implementation of the CPMF framework through the supporting prototype. It describes the architecture of the prototype. Implementation of the prototype is defined in two parts: the development of processes and their execution. For the development of processes, a graphical and a textual process editors are introduced. The transformations of process models from one phase to another is also explained in this part. The second part explains the execution support for process models using a process interpreter. Finally, the execution support for dynamic activity creation and dynamic adaptations are explained.*

5.1 Prototype Architecture

One of the major issues in software process development approaches is a lack of consistent support for the entire process lifecycle. This means that different approaches need to be followed in different phases of process lifecycle. Using multiple approaches in process development lifecycle creates issues like loss of semantics when transforming a model from one technology to another, lack of traceability, inconsistent tool support, *etc.* To overcome these issues, a single consistent approach that takes care of the process development in all its associated lifecycle phases is fruitful. We present a comparison of the related approaches and their support for associated process lifecycle phases in figure 5.1.

Business process modeling approaches are commonly used in software projects as well. We see that BPMN, does not support the development of an abstract level strategic process model without any implementation details. It offers a single process

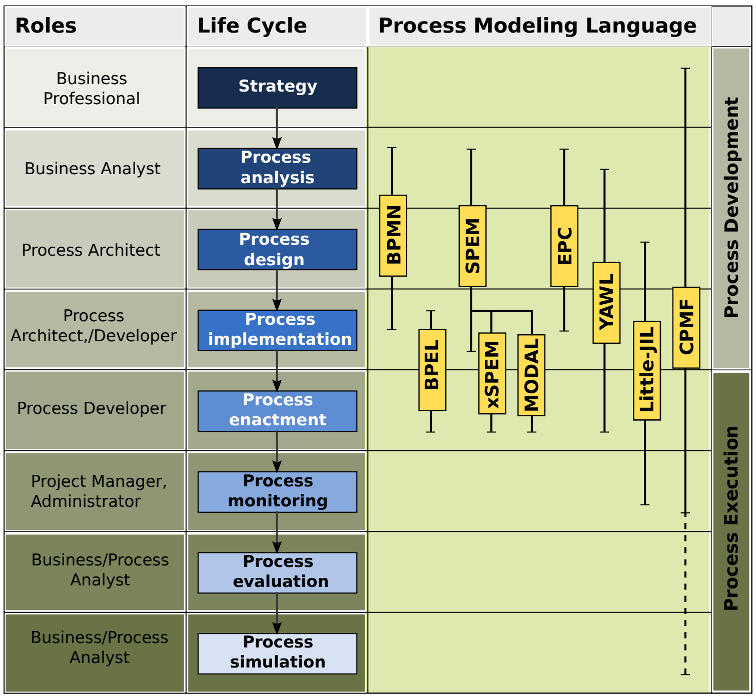


Figure 5.1 – Related process modeling approaches

modeling notation that can be used from process analysis phase to its implementation. However, for the execution, BPMN process models need to be transformed into BPEL models. This transformation allows the enactment of the modeled processes. SPEM process models suffer from the same shortcomings, even through it provides better completeness for software development processes. The lack of execution semantics in SPEM, do not allow the process models to be enacted directly. They either need to be transformed to some other approach that allows process execution like BPEL or can be mapped to the project management suits, where their execution semantics rely on the implementation choices of the process developer. Extensions to SPEM like xSPEM and MODAL provide the needed support for the execution of SPEM process models. EPCs and YAWL relatively cover more phases of process lifecycle, but without exploiting the concepts of abstraction, resulting in complex models in initial phases of process development.

Various process lifecycles define multiple different phases for process development and its execution. CPMF framework does not constrain the usage of some particular process lifecycle phases. But for the purpose of demonstration, we have chosen three phases: specification, implementation and instantiation. Each of these process lifecycle phases have a corresponding process metamodel to offer the concepts needed to develop a process model in that particular phase. A prototype implementation is provided along with the CPMF approach to help in developing the process models during the three chosen phases of development. However, other metamodels could be added to the prototype to customize the chosen set of metamodels.

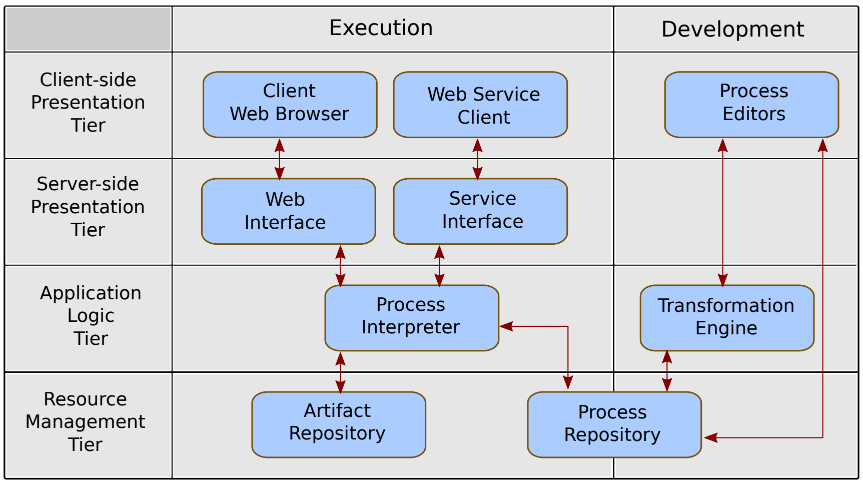


Figure 5.2 – Prototype Architecture

The tool support provided with CPMF framework allows to model the software processes in multiple phases of process lifecycle. The specification phase of process modeling allows the description of problem solving strategies for the processes in an abstract level, without polluting the model with implementation details. Details are added to the process model in each advancing phase of process lifecycle. Hence, this approach covers the first six phases of process lifecycle illustrated in figure 5.1. In this figure, a dotted line extension beyond the monitoring phase, reflects the ability of the approach to model those development phases as well, if their respective metamodels could be developed and integrated in the accompanying process modeling prototype. The approach itself allows to model the processes in any phase of process development, but the prototype is limited to first six phases for demonstration purposes.

The development of tool support for the CPMF framework has allowed us to concretize our ideas and to see the phase-wise development of process models in action. This prototype models real life processes and demonstrates the effectiveness of phase-wise modeling of software development processes. Along with the support for process modeling, this prototype is equipped with an interpreter that can interpret executable processes from the model. Processes in software development projects being distributed in nature, need to support geographically separated collaboration. This is handled through the use of service oriented architecture for the software project management dashboard. Processes also deal with a large number of artifacts and thus resource management routines need to be taken care of. An n-tier architecture is chosen for the implementation of this prototype, as shown in figure 5.2.

5.2 Process Development

5.2.1 Process Editors

The initial components of our prototype implementation are model editors that are capable of modeling the software development processes conforming to their respective metamodels, depending upon the phase of model development. For example, the specification phase provides the notions of activity definitions, abstract processes, responsibilities, etc. There are two model editors integrated within the CPMF prototype implementation: the graphical editor based on Openflexo [Openflexo 13] viewpoints and the textual editor based on Xtext [Efftinge 06]. Openflexo is an open source modeling suite that allows to develop dedicated tools for software development models. The viewpoint component in Openflexo is used to develop model editors that represent different views of a model. The motivation of using Openflexo for the development of process editor comes from the integrated software development environment that it offers. Apart from integration to the software development environment, it provides easy mechanisms to develop further different views based on a single concrete model. Thus the process models developed in the CPMF graphical editor can be used to extract different views, when needed.

Graphical models are often chosen for their ease of understandability and intuitive nature. However, if the complexity of the model grows, graphical models become cluttered and are very difficult to develop and understand. We have also developed textual editors for different phases of process development. Textual models in CPMF allow the development of complex models, where graphical models may seem verbose. Process models developed in the graphical editor allow to add additional information regarding the models through *model inspectors*. For example, the artifact specifications and the artifact lifecycle defined in an abstract artifact contract of an activity can be specified in the model inspector for the associated activity. These process models are saved as project files that contain the ontology model for the process. A process repository is associated with the model editors, which allows to store the process models. These process models can be retrieved from the repository based on the tag associated with each process.

Besides developing a process model from scratch, the model editors are also used to inject details in the existing process modeling. When the process models are transformed from one phase to another, additional structural entities and properties are added to the process model. In order to add details to these added entities, models are loaded into the model editor, which allows to enrich the models with further details. These refined process models can then again be stored into the process repository for further usage. One of the objectives of phase-wise development of process models was to support reuse. This is made possible by accessing the activity components from the repository and assembling them together to create/update a process model.

5.2.2 Transformations in Process Models

The core of the model editor is based upon the three process metamodels provided by the framework and the transformation definitions between them. A process model in specification phase is developed either from scratch or by reusing already developed processes, present in the process repository. Specification level process model is a non executable representation of the processes at an abstract level which are used for defining the structure of the process model. This process model is parsed into an xml file using the *Document Object Model* (DOM). Once a specification model is developed, it is stored in the process repository for reuse. This process model is used for the strategic decisions taken by the software project management teams. This xml representation of the process model can be bootstrapped by the process interpreter to recreate the respective objects for all the activities in the process model.

A *model transformation* is developed to transform the specification process models into their implementation counterparts. This transformation is handled by the transformation engine, developed in java. Because of the simplicity of refinement transformation, the transformation definition is written in plain java. Transformation engine maps the constructs of the specification model to the constructs of the implementation model and creates an implementation level process model. As the hierarchy of activities is considered as a specific implementation in the PImp, the hierarchical structure contained within each activity is transformed as a particular implementation of that activity.

'Slots' are created in the process implementation model to add other implementation level details. These details are entered manually, using the process editors. These details may vary from activity properties to adding the activity and associated artifact state-machines. Implementation level process models can also be stored in the process repository for further reuse. Traceability links are maintained in all the process models after specification phase. Each activity definition in the implementation level process model can be traced back to its counterpart in the previous phase. These traceability links are also maintained in the future phases of process development.

Implementation process models can be transformed into the instantiation process models by the transformation engine. Instantiation level process models are executable and need further instantiation level details to be able to execute. 'Slots' are again created in the process instantiation model that require the instantiation level details. These details are entered manually in the instantiation model. Multiple activity implementations are possible for each activity definition in the implementation level model. These slots are created in all of the activity implementations and are carried on to the instantiation model. However, the instantiation details are only entered into the chosen instantiation activities for each activity definition. These instantiation activities can replace each other. This replacement of instantiation activities is handled as process adaptation by the process interpreter during execution.

5.3 Process Execution

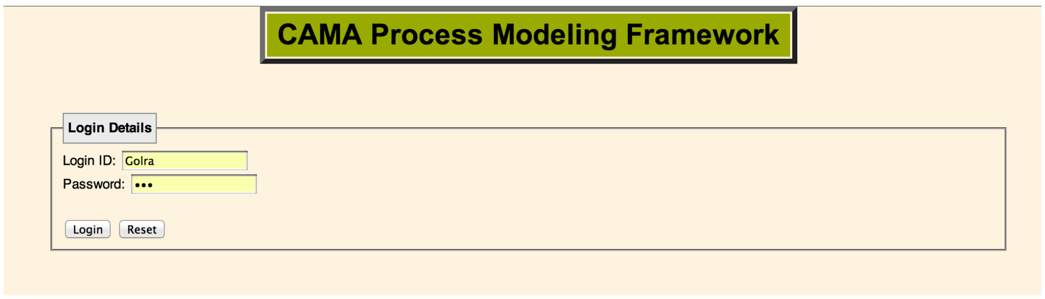
5.3.1 Process Enactment

The process interpreter can retrieve the executable process models from the repository. It then bootstraps the process models and creates the respective runtime objects for the activities present in the process model. The process interpreter incorporates an event management system that is responsible for the runtime execution of these activities. Our choice of relying on events for the concrete process model was motivated by the intentions of adding reactivity to the process models. This choice of event based implementation favors to decouple the activities as well. The process interpreter is also loaded with activity factory, that allows runtime creation of activity implementations. Underlying bi-layered process metamodel allows us to add, remove or substitute instantiation activities in an instance process model, at runtime.

Geographical distribution is an inherent nature of software development processes, where stakeholders need to collaborate in order to execute the processes. Java servlets are used to develop a web interface for the process interpreter. The web interface presents a project management dashboard, which is used to monitor the execution of the software development processes. The project management dashboard presents a customized interface for each role in the software development project. The project manager has access to all the activities in the project for planning and monitoring purposes. States of all activities and artifacts are displayed in the project management dashboard to monitor the advancement of the executing processes. Each role has access to the details of the activities associated with it.

In order to cater the needs of resource management, an artifact repository is also added to the prototype. Artifact versioning is supported by the artifact repository. The details of artifact versioning are kept hidden from the developers. Developers are responsible to develop the artifacts and upload them to the software project management dashboard. The project management dashboard is linked with the artifact repository and handles the versioning of artifacts. It also takes care of the locking mechanism for each artifact and allows access for the artifacts to the concerned roles of the activities only. Every activity has access to the associated guidelines and the input artifacts. Rights are associated to each action associated to the artifact and the activities. Only the roles that have access to a particular action for an activity or an artifact are allowed to carry out the task.

The project management dashboard is implemented in a way that it is based on the contracts of the activities. The user interface adapts itself according to the role, associated actions and state of the activity. Figure 5.3 presents different screen-shots of the project management dashboard and each snapshot reflects a different aspect: a) Each actor can log in to the project management dashboard and interact with the activities assigned to him. b) Project management dashboard is capable to handling multiple projects at a time. An actor may be working on multiple projects within an organization. c) Provided artifacts for an activity can be uploaded through the project management dashboard. d) The project management dashboard reacts to the

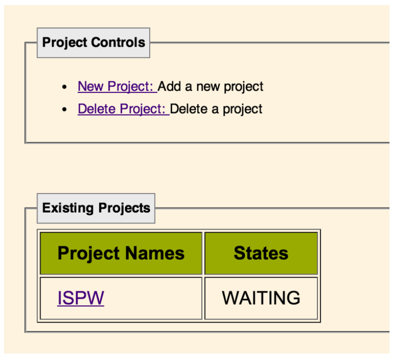

CAMA Process Modeling Framework

Login Details

Login ID: Golra

Password: •••

a) Authentication for every actor and process owner

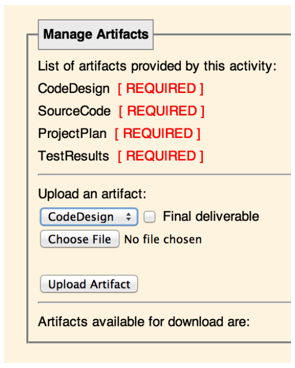

Project Controls

- New Project: Add a new project
- Delete Project: Delete a project

Existing Projects

Project Names	States
ISPW	WAITING

b) Project Management Dashboard


Manage Artifacts

List of artifacts provided by this activity:

CodeDesign [REQUIRED]
SourceCode [REQUIRED]
ProjectPlan [REQUIRED]
TestResults [REQUIRED]

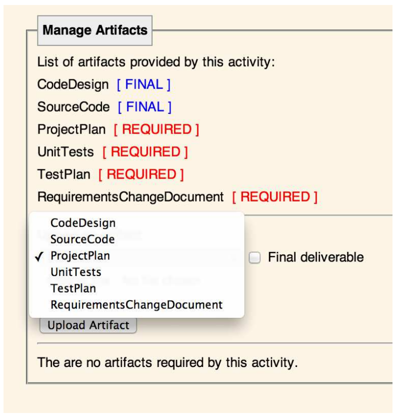
Upload an artifact:

CodeDesign ☐ Final deliverable

Choose File No file chosen

Artifacts available for download are:

c) Activity (no artifact provided)


Manage Artifacts

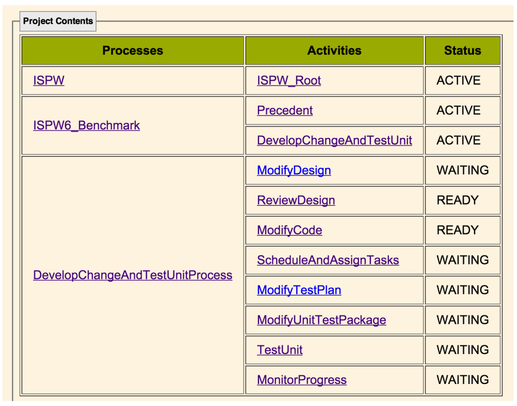
List of artifacts provided by this activity:

CodeDesign [FINAL]
SourceCode [FINAL]
ProjectPlan [REQUIRED]
UnitTests [REQUIRED]
TestPlan [REQUIRED]
RequirementsChangeDocument [REQUIRED]

☐ Final deliverable

The are no artifacts required by this activity.

d) Activity (two artifacts provided)


Project Contents

Processes	Activities	Status
ISPW	ISPW_Root	ACTIVE
ISPW6_Benchmark	Precedent	ACTIVE
	DevelopChangeAndTestUnit	ACTIVE
DevelopChangeAndTestUnitProcess	ModifyDesign	WAITING
	ReviewDesign	READY
	ModifyCode	READY
	ScheduleAndAssignTasks	WAITING
	ModifyTestPlan	WAITING
	ModifyUnitTestPackage	WAITING
	TestUnit	WAITING
	MonitorProgress	WAITING

e) Activities with corresponding states

Figure 5.3 – Project Management Dashboard

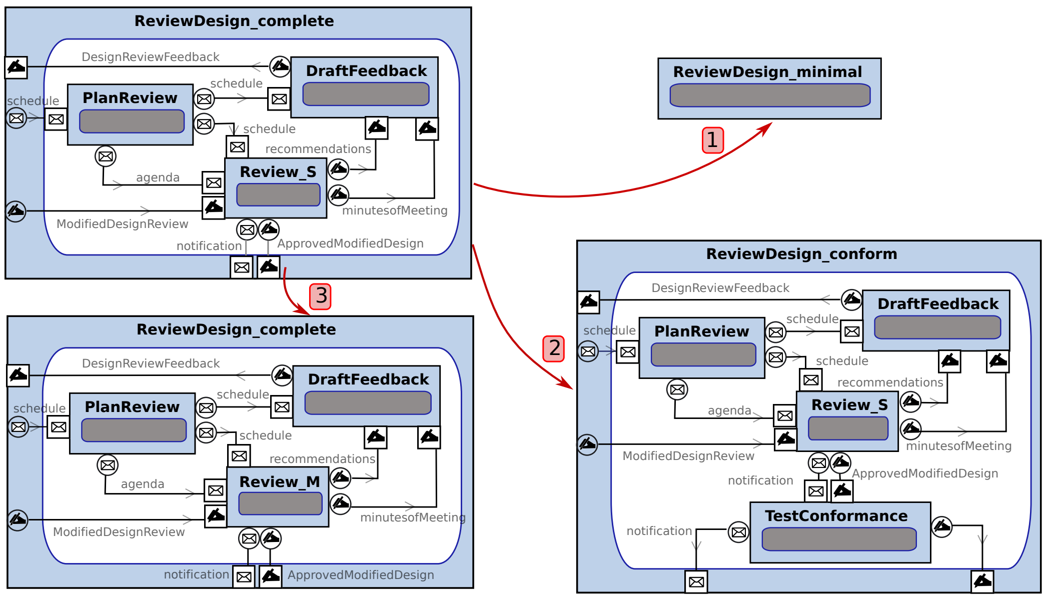


Figure 5.4 – Runtime adaptation

change of artifact state. It adapts the interface accordingly. e) An actor is presented with the state of the associated activities and other guidelines, properties associated to the activities.

5.3.2 Execution dynamics

Requirements volatility is an issue faced by software development communities for long. These changes in software requirements trigger a change in the specification and design of the system under development. They also affect the processes being followed to develop such systems. There are other motivations like momentary customizations and evolutionary changes that demand a process system to allow a certain level of flexibility at runtime to support manipulations [Mutschler 08, van der Aalst 00]. CPMF framework offers the required flexibility to evolve, update, improve, or customize a running process by adding, removing or substituting activities in it.

We have used the ISPW-6 benchmark (section 4.1.2) as the running case study to explain the CPMF process models in the previous chapter. For the purpose of explaining the runtime support for activities during their execution, we will take three scenarios from the same benchmark. The *ReviewDesign (RD)* activity from the ISPW-6 benchmark problem is a composite activity implemented using three sub-activities, as shown in Figure 5.4. The dependencies between the activities are shown explicitly in the figure for the purpose of understandability only. The first scenario deals with a temporary change in the process model under the current execution scheme. In the current model, the RD activity has a feedback loop to the *ModifyDesign* activity. Let us imagine a scenario where the review has already been carried out twice and the current design is almost perfect except that it needs some slight changes. Thus the review committee decides to approve it with these changes in a way that the next review should not be that elaborate. Thus the review activity is customized in a way that only one person reviews the design for the said changes and sends the acceptance notification to *MonitorProgress* activity. The actual RD activity is implemented using three sub-activities, where as the customized RD activity would be a primitive activity. In this scenario the pre-adaptation activity is composite, whereas the post-adaptation activity is primitive.

In a second scenario, we imagine a perfective process improvement, where the RD activity is changed in a way that a new sub-activity is added to it. This new sub-activity verifies the conformance of design to the adopted standard. This new sub-activity is performed by an external consultant. The new RD activity remains a composite activity in this scenario, but its sub-assembly is updated. Both pre-adaptation and post-adaptation activities in this scenario are composite.

The third scenario illustrates the change of one sub activity from RD activity implementation. In incremental development lifecycles, the MD activity might not give the complete design for review, it can rather give the design of a module for the review. In such case when the design review event is triggered by the MD activity, the preconditions of the RD activity are verified. RD activity triggers the *PlanReview* (PR) activity, which is an automatic activity. This activity checks the calendars of the participants and the availability of the meeting rooms to schedule a *Review_S* (RS) activity. Based on the *scope* property, it triggers an adaptation in the RS activity. One implementation of RS activity targets the review of module design, which is a short review, while the other is a comprehensive review that takes into account the effect of modified design on complete system. In this scenario, both the pre-adaptation and post-adaptation activities are primitive.

5.3.2.1 Dynamic Creation

In order to induce a certain level of automation in the framework, we argue that the possibility of dynamic activity creation is of high value. By dynamic activity creation, we mean that activities can be created at runtime, either manually or (semi-) automatically. This allows to evolve the process model at runtime by adding new activities to it, which were not planned at the specification and implementation time. For processes that run for years and can not be brought down for evolution, we need a possibility to create new activities and add them to the process.

An activity Factory has been designed for the process model at runtime. This module is integrated in the process interpreter and during execution a call to it returns an activity implementation. For the current version of the tool implementation, we do not support runtime creation of activity definitions. An activity definition should already be present at the abstract level to create an instantiation activity that conforms to it. The contracts of the instantiation activity need to conform to that of activity definitions. In order to create an instantiation activity, we need to specify its activity definition. Development of a contract for activity implementations that does not conform to its counterpart at activity definition is restricted by the editor. Thus the framework ensures that all concrete processes conform to the abstract process.

For manual creation of activities, the process designer is responsible to create the activities at the runtime. By calling the activity factory and specifying the required features manually, instance activities are created. For semi-automatic creation of instance activities, a new activity implementation can be created by the system based on the contracts already specified at the abstract level. Only primitive activities can be created (manually or) semi-automatically by the framework. Composite activities can be created manually by composing primitive activities. Primitive activity creation can be semi-automatic because the designer needs to trigger this creation, however he does not need to specify the concrete contracts of the activity manually. The concrete contracts of the activity are developed by the system, based on the default state-machines of the artifacts at the abstract level. An example of this type of activity creation would be an alternative primitive implementation of *Review_S* sub-activity of the *ReviewDesign_complete* activity. Let us name this alternative implementation as *Review_S2* that conforms to the same activity definition, *Review_def*. In this case, the contracts of *Review_def* are specified at the abstract level, and at runtime the alternative activity implementation, *Review_S2*, is created based on those contracts (having default state-machines for each contract). This implementation of the review activity can accept review schedule, and it can create events for notifications, forwarding approved design and recommendations and choose any means like emails or hand carried hard copies. The role of this activity is also chosen, based on the responsibility assignment matrix, where a senior design engineer can perform this activity.

5.3.2.2 Dynamic Adaptation

CPMF allows the specification of the software development processes in the first phase, which are then refined into concrete processes with implementation details in its second phase. The process implementation model defines multiple activity implementations for each activity definition defined at the abstract level. Process Implementation metamodel introduces a certain degree of variability in the model that depends on the process requirements, contextual fluctuations and runtime criticality of the process modeling system. Finally in its last phase, a set of activity implementations is chosen for injecting the instantiation level details and the process model is transformed into an executable model. Such a strategy defers variability binding until runtime. During the course of execution, modifications to the structure and behavior of concrete process can be carried out, such as adding, removing, substituting its instantiation activities. It is worthwhile to note that we are not targeting towards an adaptive process model where the abstract processes adapt at runtime. We have limited our scope in this framework to provide dynamic adaptations for the concrete processes only. A future perspective of this thesis is to provide dynamic adaptations to the abstract level of the process as well. This will overcome the restriction to adapt an activity to other implementation alternatives only. The process designer would be

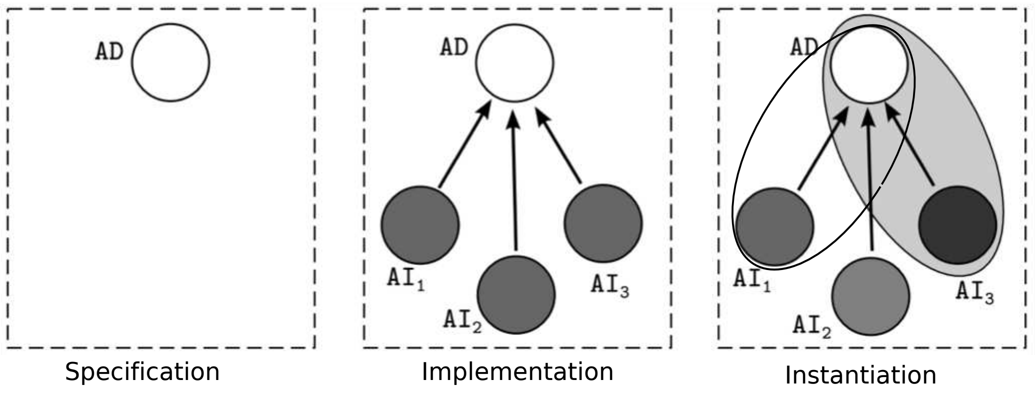


Figure 5.5 – Activity implementation variants

able to add new activities from scratch by adding both activity definitions and the corresponding activity implementations.

Variation is introduced in the software process modeling in its implementation phase, where multiple activity implementations can be developed for each activity definition, as shown in figure 5.5. All these activity implementations conforming to a single activity definition need to conform to its contracts. The contracts at the concrete level contain events that map to the artifact specifications at the abstract level. This mapping is used to ensure the conformance of activity contracts. Conformance of artifacts to their specifications and roles to responsibilities is also ensured by a mapping between them. The CPMF framework guarantees the conformity of an activity implementation to its definition through restricting the development of non-conforming implementations. The number of possible configurations of an executable process model depends on the total number of implementation variants for all activity definition.

Variability points in a process model, refer to the situation where multiple activity implementations are available for execution and one of them is chosen, based on the satisfaction of its precondition. These variability points are responsible for triggering the adaptation. Preconditions are used to constrain the execution of activity implementations to the specific context configuration. Besides this, preconditions also allow to catch the unexpected behavior of the process. In such situations, a precondition acts as a variability point and triggers a dynamic runtime adaptation. This concept of variability point is not part of the metamodel, rather it is an implementation choice for the prototype. Runtime adaptations can also be triggered by human decisions for various intentions like process improvements or runtime process composition. The project management interface for the process owner, allows to adapt the processes. This interface presents different implementation alternatives of an activity implementation. The process administrator can choose to replace an activity implementation with another.

Post-conditions of an activity implementation model the impact of its execution on the system configuration. A system configuration is a snapshot of all the activities

```
Activity ReviewDesign_minimal(ReviewDesign RD_complete)
              implements ReviewDesign_def{
  description ("This activity is responsible for the review
                of the modified design...")
  properties [
   reviewer = RD_complete.Review.reviewLeader;
   scope = RD_complete.PlanReview.scope;
   iteration = RD_complete.PlanReview.iteration;
   approval = RD_complete.DraftFeedback.approval;
   -
   -
   ]
  External Contracts [
   Req event ModifiedDesignReview_R refersTo ReviewDesign_R
   Prv event ReviewDesignFeedback_P refersTo DesignFeedback_P
   ]
  roles [
   ProjectManager refersTo Authority
   DesignEngineer refersTo Responsible
   ]
}
```

Listing 1 – State mapping scenario 1

in the running system along with their properties. We can use these post-conditions to reason about the execution outcomes of a process. The execution of the process can be abstracted as a state machine, where states are the possible execution configurations and the transitions are the possible adaptations: human triggered or dynamically triggered. A complete specification of state machine can help the process designer to perform various tests and validations before the actual implementation.

The adaptation of activities might need the creation of a new instantiation activity in case its corresponding activity implementation was not created in the implementation phase. If the instantiation activity already exists, it is simply chosen to replace the current instantiation activity. In both these cases, the state and properties of the current activity implementation need to be transferred to the new instantiation activity that would replace it. In the process of creation, a mapping towards the properties and state of the old instantiation activity is established through the use of a constructor. The limitation of this method is its hard-coded mapping to each available instantiation activity. In order to overcome this limitation, we also offer an interactive state transfer mechanism where the mapping between the old and new instantiation activity are presented to the process owner on web interface. Process owner can chose to map the properties that need to be kept intact for state transfer.

Let us examine the runtime adaptation in action by looking into each scenario, presented in the last section. The first scenario substitutes the *ReviewDesign_ complete* instantiation activity of *DesignReview* activity definition with a another implementation, *ReviewDesign_ minimal*. We assume that only the *ReviewDesign_ complete* instantiation activity was available at runtime and we did not have access to an instance of *ReviewDesign_ minimal* activity. In this scenario, the *ReviewDesign_ complete* ac-

```
RD_conform.Review.reviewer = RD_complete.Review.reviewLeader;
RD_conform.PlanReview.scope = RD_complete.PlanReview.scope;
RD_conform.PlanReview.iteration = RD_complete.PlanReview.iteration;
RD_conform.DraftFeedback.approval = RD_complete.DraftFeedback.approval;
```

Listing 2 – State mapping scenario 2

tivity has already been executed several times in a feedback loop with the *ModifyDesign* activity. At a certain point in time, this adaptation is triggered by a human decision. In this case a new instantiation activity needs to be created at runtime. As this activity is human triggered, the responsible role for the activity needs to create the *ReviewDesign_minimal* activity. This activity is created using the state update mechanism, thus the properties and state of the *ReviewDesign_complete* activity are transferred to it. The hard-coded mapping is specified in the constructor of the *ReviewDesign_minimal*, a part of which is listed in Listing 1. The state of the *ReviewDesign_minimal* is recovered from all the sub-activities of the *ReviewDesign_complete* instantiation activity. *ReviewDesign_minimal* being a primitive activity does not have any sub-activities and thus has to carry these properties. This adaptation substitutes a composite instantiation activity with a primitive one.

For the second scenario, a new sub-activity is added to the existing *ReviewDesign_complete* activity. This adaptation is also triggered by a human decision for the purpose of process improvement. The first step in order to continue this adaptation is to create the *TestConformance* activity at runtime. We assume that the activity definition for this activity is already available at the abstract level. This activity is created using the activity factory provided by the framework. This activity is created from scratch and does not need to import any sort of state from any other activity. However the placement of this activity in the *ReviewDesign* activity needs an adaptation on part of the *ReviewDesign_complete* instantiation activity. A new *ReviewDesign_conform* activity is created which imports the state for *PlanReview*, *Review_S* and *DraftFeedback* sub-activities from *ReviewDesign_complete*, using the static state update, as shown in Listing 2. This adaptation is from a composite activity to another composite activity which has got an additional sub-activity.

The third scenario substitutes a sub-activity from the *ReviewDesign_complete* activity with another, where the activity implementations for both these instantiation activities were developed at the implementation phase. Figure 5.6 shows that a modify review event is triggered by *ModifyDesign* activity. This event is received by the *ReviewDesign_complete* activity which delegates it to its *PlanReview* Activity. There is a tight integration between the Openflexo development environment and the process interpreter. PlanReview activity is an automatic activity and would automatically plan the review based on the calendars of the participants and availability of the resources. The scope property of the *PlanReview* is calculated from the type of Design offered by the *ModifyDesign* activity. The two implementations of Review activity are *ReviewSystem* and *ReviewModule* instantiation activities. Which of these instantiation activities would be used for the current iteration depends upon the scope property of the *PlanReview*. The preconditions of the two implementations specify

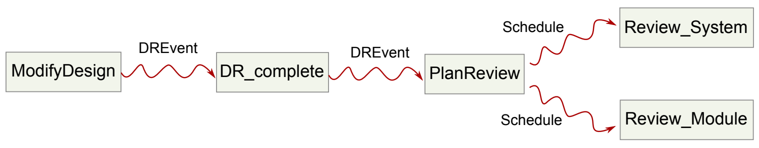


Figure 5.6 – Dynamic adaptation

if they can be executed with the current value of scope. If the current instantiation activity in the system satisfies the condition, it is executed and no adaptation is required. However if the precondition of the current implementation is not satisfied then it can trigger an adaptation. The process interpreter would look for an alternative instantiation activity that conforms to the same definition to run this adaptation. This adaptation would substitute the current instantiation activity implementing the Review activity definition with the one whose precondition is satisfied.

In this case, let us assume that *ReviewSystem* instantiation activity was bound for the current execution and at runtime the *ModifyDesign* instantiation activity offered the updated design of a single module. In this case, the event would be delegated by the *ReviewDesign* activity to the *PlanReview* activity which would automatically plan the review and send emails to the concerned roles for a confirmation. The scope of the plan would be set to module review. The precondition of the *ReviewSystem* activity would not be satisfiable and then it would trigger an adaptation. This adaptation would access the *ReviewModule* instantiation activity and a state update would be done on it to copy the state of the *ReviewSystem* instantiation activity to it. Finally the instantiation activities would be replaced and the system configuration would be updated.

5.4 Implementation Summary

One of the problems with existing process modeling approaches is that they support a small part of the complete process development lifecycle. They often tend to focus on either implementation or instantiation phases of process development. Process designers need a different approach for process specification, transform them to implementation approaches and then further transform them to instantiation approaches for making the process model executable. Some of the approaches cover both implementation and instantiation phases, but a transformation is needed to support complete process development lifecycle. These transformations from one approach to another result in semantic losses, because of inconsistent process modeling platforms. CPMF provides an implementation prototype that can be used to develop process models for specification level, as shown in figure 5.7. These process models are developed using either a graphical process editor provided with the prototype or through the textual domain specific language. One of more process specification models are then transformed to the process implementation models using a transformation engine, provided with the prototype. This transformation engine transforms the pro-

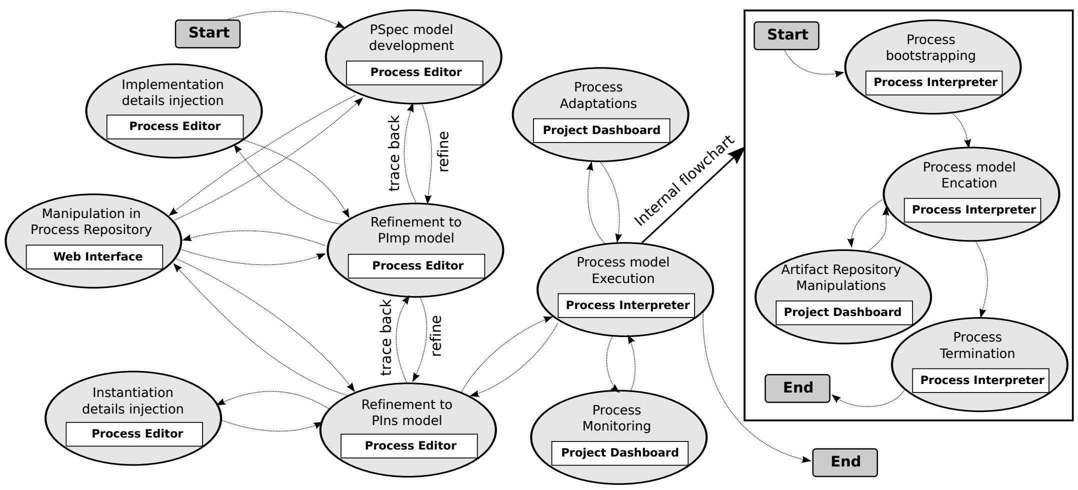


Figure 5.7 – Process Implementation flowchart

cesses from specification to implementation and from implementation to instantiation phases.

Once the process models are refined to instantiation phase, they become executable. These executable process models can be loaded into the process interpreter provided in the implementation prototype. The process interpreter bootstraps the process model and executes it. An embedded web-server (developed using Jetty) allows to setup a web interface for interacting with executing processes. This web interface is called a *Project Management Dashboard*. A process owner can access all the activities of a process that he/she owns. Actors playing specific roles associated with the activities can access those activities only. Once the artifacts required by an activity are available, the state of the activity changes to 'ready', in the default activity statechart. The associated actor of the activity can download the required artifacts, do the processing and upload the developed artifacts through this web interface. Process owner can adapt the activities to one of the implementation alternatives already developed from implementation phase. Adaptation of the activities in an executing process model needs to take care of the transfer of state between the old and the new activity implementation. This is handled by the interpreter using either hard-coded mapping through activity implementation constructors or by presenting the properties of the two implementations to the process owner, which can link them together to a transfer of state.

Part III

Evaluation of the Framework

Chapter 6

Case Study

Contents

Abstract *- This chapter presents the application of CPMF framework on a pseudo-real case study to elaborate and justify the claims made in this thesis. The scenario chosen for this case study is termed as pseudo-real for the reasons that the original process is modified to illustrate the key concepts of this approach. The scenario is explained in the first section. Second section presents the implementation of the scenario processes in CPMF framework. Finally, this chapter concludes with a discussion on the implementation of this case study.*

6.1 Case Study Scenario

Case studies are a common strategy to elaborate and evaluate the proposed research methodologies in many fields. This can be attributed to its strength in helping the investigators to understand complex inter-relationships between the varying claims in their propositions. They rely on the applicative evidence of already developed theoretical propositions. Successful applications demonstrate the feasibility of the proposed methodologies.

In order to explore the applicability of refinement based process modeling methodology and to uncover improvement opportunities for this method, we have conducted this case study. It involves the implementation of pseudo-real processes from a software enterprise using multiple models. Each of these process models are specific to a particular development phase of software process development lifecycle. It involves a usage of the supporting tool implementation provided with this research thesis. The process development modules of the prototype are used to develop the processes of this scenario and the process execution modules are used to execute them.

The implementation of this case study starts with the development of process models from specification level. It illustrates the refinement of the process model to the later phases of process development. Reuse of existing process elements from process repository, compliance of process elements to the chosen standards and the contractual interactions between the process components are also presented through this case study implementation. Enactment of the process model that has been developed through multiple refinements validates its executability. Runtime adaptations to the executing processes are presented to discuss the dynamic nature of the process modeling approach. Finally, the possibilities of monitoring the processes in execution are presented for this scenario.

6.1.1 Background

The process covered in this case study concerns the testing phase of software development. A software enterprise needs to develop the testing phase process for a particular software development project, *AlphaSystem*. This enterprise has its own organizational standards to develop the testing processes. We assume in this case study that the organizational standard was already developed using CPMF framework (for previous projects handled by this enterprise). Apart from its own organizational standard, this organization also wants to follow the ISO/IEC 29119 standard [IEEE 13] for software testing. Consequently, the processes being implemented for this particular software project need to comply with both these standards (organizational and ISO/IEC 29119).

Testing process is developed for the specification phase and stepwise refined till the instantiation phase. Reuse of already existing processes in the process repositories for the development of current testing process is also taken into account. Once the executable model is developed, it is executed on the process interpreter. Execution of the testing phase processes is monitored through the project management interface. Runtime adaptations are carried out on the testing process for *AlphaSystem*.

The ISPW-scenario used in chapter 4 and 5 to illustrate the CPMF process modeling approach is a standard benchmark, which is an abstract representation of the real life processes in the software industry. On the other hand, the testing process chosen for this case study is a real life process carried out in the testing phase of a software enterprise. This software process is adapted for a better demonstration of the key problem areas identified in the current process modeling approaches. The intent remains to come up with a complete model of the processes that can represent the way a software development organization typically works. The names of the project, actors and the software enterprise are fictitious.

This case study would enable an appropriate guidance for the future users of the CPMF approach. This motivation provides a provision for the case study to focus on the main propositions of the approach and eases the way for the acceptance of this methodology. The next section describes the scenario presented by the case study and then section 6.1.3 details the link between the research questions of this thesis and the case study.

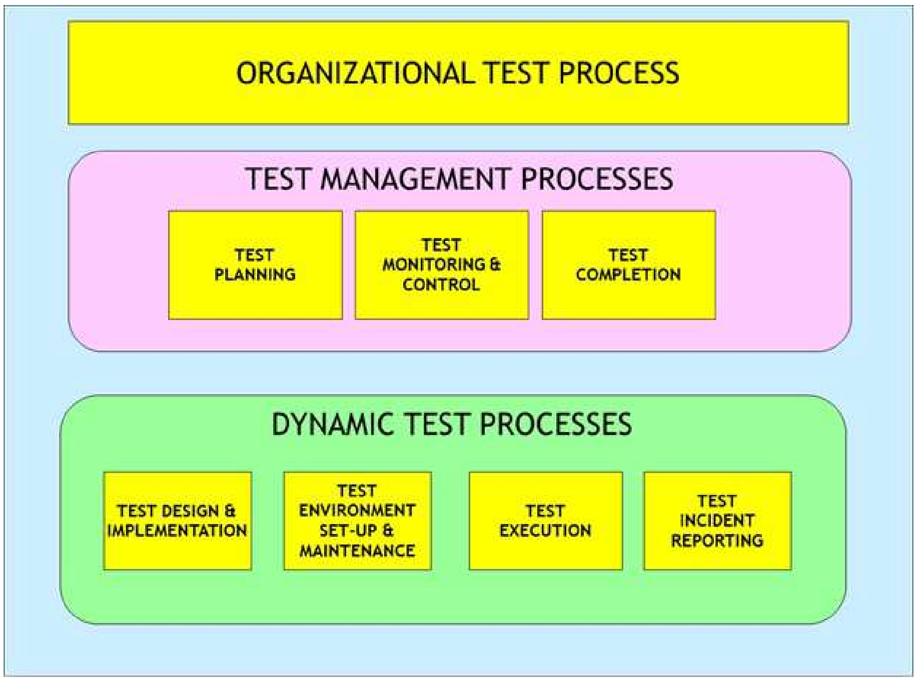

ORGANIZATIONAL TEST PROCESS
TEST MANAGEMENT PROCESSES
TEST PLANNING
TEST MONITORING & CONTROL
TEST COMPLETION
DYNAMIC TEST PROCESSES
TEST DESIGN & IMPLEMENTATION
TEST ENVIRONMENT SET-UP & MAINTENANCE
TEST EXECUTION
TEST INCIDENT REPORTING

Figure 6.1 – Test Process ISO-29119-2 [IEEE 13]

6.1.2 Scenario

A software enterprise, *TB-Enterprise* is developing a software system for its clients. The software system is named as *AlphaSystem*. During the course of this development project, this enterprise needs to go through the testing phase of the software system under development (among other software development phases). The development of the processes for the *AlphaSystem* project takes place before the actual execution of these processes. *TB-Enterprise* has developed its own organizational standards for the software development projects. That organizational standard covers the testing process as well, so we take into account the testing process part of the standard only. However, for the *AlphaSystem* project, the client wants *TB-Enterprise* to follow the ISO/IEC/IEEE 29119-2 standard. Thus, this enterprise is planning to follow both the standards to develop the testing process for *AlphaSystem*.

The 29119-2 international standard presents the software testing process in multiple layers as shown in the figure 6.1. The first level presents the *organizational test process*, which covers the development of organizational test process specification, its use and its updations. *TB-Enterprise* follows the first level and develops the test process specification using a specification process model of CPMF approach. This model will further be implemented, used and finally stored in the process repository for any updation, if necessary. The second level of the standard presents the *test management process*, which defines the activities for managing the process through *planning*, *monitoring & control* and *completion* activities. Finally the third level presents the *dynamic test process*, which focuses on the design and development of the tests. All

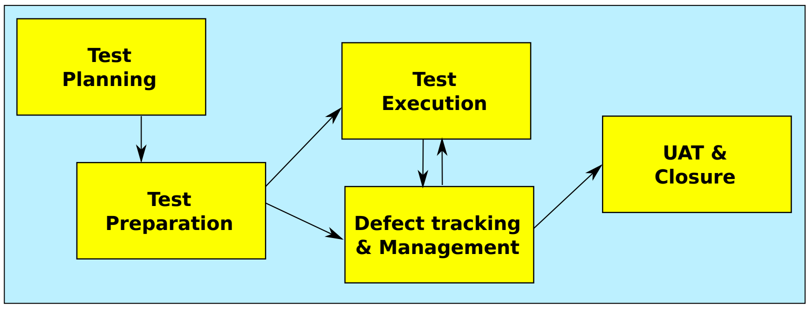


Figure 6.2 – Test Process Organizational Standard

the activities presented in the second and third level of this abstract process model are composite activities which have further sub-activities to develop a fine-grain process model. Standards are developed in CPMF framework as specification process models. A specification process model for 29119-2 standard is developed by *TB-Enterprise* for this project and stored in the process repository for possible future reuse.

The organizational standard for software testing process is presented through five main activities. *Test planning* is the first activity followed by the *test preparation* activity. These activities are then followed by two activities in parallel: *test execution* and *defect tracking & management*. The *test execution* activity contains the sub-activities that are responsible for executing different tests like smoke tests, regression tests, integration tests and system tests. The final activity is *user acceptance testing & closure*, which ends the testing process, when the end user/client accepts the *AlphaSystem*. We have assumed in this case study that this organizational standard was already developed under CPMF framework by *TB-Enterprise* for some other projects. The specification process model of this organizational standard, *TB-Enterprise testing process*, is retrieved from the process repository and used for the development of the test process for the *AlphaSystem* project.

Both specification process models (each modeling a different standard) are refined into a single process implementation model for *AlphaSystem*. The abstract level of this implementation process model presents a workflow kind of model that complies with both the standards. The activity definitions present in the abstract level of this model are implemented by the concrete activity implementations. Some of the activity definitions in this process model will be implemented by multiple activity implementations to allow a degree of process variation. This case study also reuses some of the activity implementations already developed for prior projects. They are also retrieved from the process repository and reused.

Finally, the implementation process model is refined to the instantiation process model. This executable process model is executed by the process interpreter. A web interface is used to monitor and control the executing processes. Artifacts created during the execution of the process are stored and retrieved from an artifact repository associated with the process interpreter. This artifact repository keeps these artifacts under version control and access to them is allowed to the associated roles only. During the execution of the process model, it can be adapted by replacing the instantiation activities with other instantiation activities already transformed from alternative activity implementations.

6.1.3 Questions & propositions

The research questions targeted by this thesis are presented in chapter 1. We link those research questions to the different aspects of this case study before presenting its implementation. This way, the readers may connect the research problems to the methods with which they are dealt in CPMF approach.

— **RQ-1: Contractual interactions** CPMF process modeling approach defines the contracts for every activity in the process model, either at the abstract level or at the concrete level. All the interactions between the activities in the testing process of this case study are based on the contracts of the interacting activities. Contractual interactions are implemented in all three phases of process development.

— **RQ-2: Process reuse** We have assumed in this case study that the organizational standard for software testing process was developed by the *TB-Enterprise* as a specification level process model for some earlier project. Because of the abstract nature of specification level process model, it can be reused in a broader scope. The reuse of concrete level activities will be demonstrated by reusing the activity implementations from the process repository.

— **RQ-3: Data-flow vs Control-flow** The data-flow and the control-flow of the activities in the testing process are modeled at different layers by CPMF methodology. Implementation and instantiation phase process models use a bi-layered approach, where data-flow is modeled at the abstract level. The control flow of the *AlphaSystem* testing process during execution (in the interpreter) follows the concrete level model, which is developed in implementation phase and further refined in instantiation phase. The actual data-flow during execution is also handed by the control-flow events at the concrete level that map to their respective artifacts.

— **RQ-4: Compliance to standard(s)** *TB-Enterprise* has its own organizational standard for software testing. For *AlphaSystem* project, it needs to follow the ISO/IEC/IEEE-29119-2 standard as well. Thus this case study demonstrates the methods for complying to multiple standards for developing a process model.

— **RQ-5: Backward traceability** The testing process for *AlphaSystem* complies to two standards, where each of these standards is modeled as a specification process model. Activities at implementation and instantiation phase of the testing process can be traced back to their counterparts in previous phase models. In this case study, each activity (in later phase models) may be traced back to either or both ISO standard and the organizational standard.

— **RQ-6: Process/software development automation** A level of automation for the testing process development is demonstrated through the use of transformation for process refinements. The automation of testing process execution is illustrated through the capability to invoke concerned tools by some activities of this process.

6.2 Case Study Implementation

We are going to discuss the details regarding the implementation of this case study in this section. These details are discussed in a manner that we start from the specification model till the final execution of the instance process model, highlighting the key propositions of this thesis.

6.2.1 Compliance to multiple standards

It is assumed in the organizational standard of *TB-Enterprise* was developed as a specification process model and stored in the process repository. In order to develop the implementations of the *AlphaSystem* process model, this specification model is retrieved from the process repository. Process models are stored in the process repository along with different tags associated to them *e.g.* their level of abstraction (*i.e.* specification, implementation or instantiation), their objectives, *etc.* These tags allow to search process models in the process repository. Once a process model corresponding to the process requirements is found, it can be retrieved and tailored (if necessary).

TB-Enterprise Test Process is the specification level process that is retrieved from the process repository. It is an abstract level process specification model for the testing phase of software development and does not require any tailoring for our current software project. It is composed of five composite activities: *Test planning*, *Test preparation*, *Test execution*, *Defect tracking & management* and *UAT & closure*, as illustrated in the figure 6.3 . This testing process assumes that unit testing is the responsibility of the development team which should be carried out before sending the application for this testing process. Hence, unit tests are not part of the this testing process. *Test planning* is the first activity to be performed in this process and is responsible for developing a *Test plan.* It requires the *Project management plan* (PMP), *Software design document* (SDD) and *Software requirement document* (SRD). It contains further sub-activities for analyzing and planning the process, identifying the test strategy for the software project and finally the development of the test scenarios. It is performed by *Quality assurance lead* and *Testers.* Once this activity is complete, it provides the *Test plan* (TP) and the *Test scenarios* (TS).

Test preparation is an activity that depends on the *Test planning* activity. This dependency is highlighted through the binding between the two activities for TP and TS contracts, where *Test preparation* activity requires these two artifacts. Apart from these two artifacts, this activity also requires SRD, *Software development files* (SDF) and *Traceability matrix* (TM). TM is an optional contract for this activity which is used for the subsequent iterations of the activity, for updating the associated

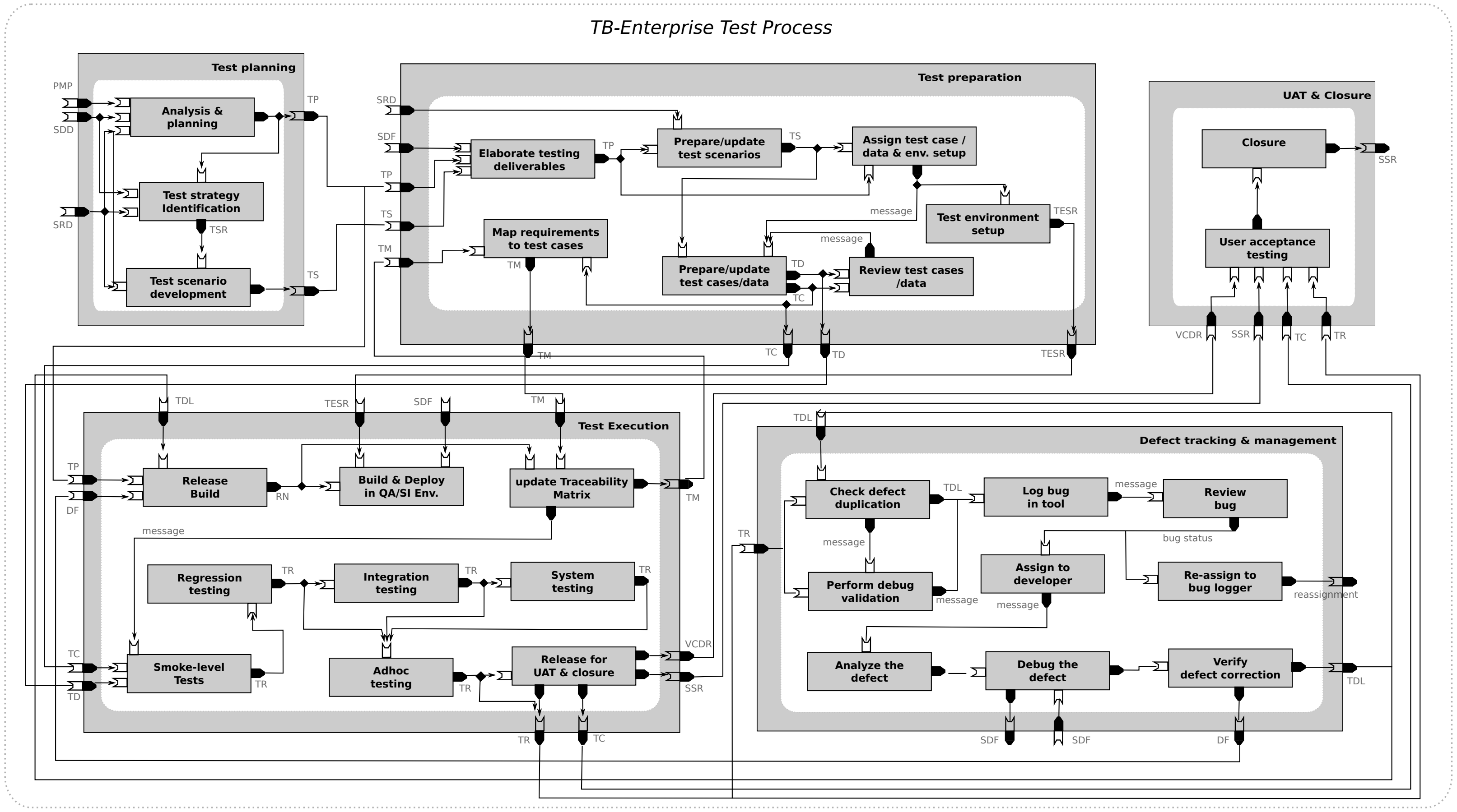


Figure 6.3 – Organizational standard for testing process

artifacts. This activity has multiple sub-activities that are responsible for developing *Test data* (TD), *Test cases* (TC), TM and for setting up the test environment. A *Test environment setup report* (TESR) is provided by this activity for the details relating to the test environment. This activity is performed by the *Quality assurance lead* (QA-L) and the *Testing team*(Ts).

Test execution follows the *Test preparation* activity and is responsible for the execution of different tests on the software system under development. It requires TP from the *Test planning* activity, hence creating a dependency between them. TD, TC, TM and TESR are also required by this activity, which it acquires from the *Test preparation* activity. Subsequent iterations of this activity requires *Test defects log* (TDL) and *Defects fixed* (DF). *Technical lead* is responsible for releasing the build. The application is then deployed in the quality assurance/ system integration environment, which allows the execution of the test. Smoke-level tests and regression tests are performed for qualifying the build for further tests and ensuring that no unwanted changes were introduced in the system after modifications through multiple iterations. Functionality, GUI, usability, security and database test cases are executed for Integration testing. Volumes, compatibility and load/performance test cases are executed for the System testing. Ad hoc testing finalizes the *Test results* from other tests and forwards it for UAT release if they are successful or to the *Defect tracking & management* activity if the tests are failed. If the tests are successful, *Verified & closed defect report* (VCDR) and *Status/summary report* (SSR) are provided by this activity. *Technical lead*, *Quality assurance lead* and testing team are responsible for *Test execution* activity.

Defect tracking & management is an activity that executes in loop with the *Test execution* activity, till all the test are successful. This activity logs the defects, after ensuring that it is not already present in the log. It also verifies the earlier debug validations. Defects are reviewed and assigned to the development team for debugging. In case any defect is invalid or is a 'known issue', it is re-assigned to the defect logger. Finally the defect is analyzed, debugged and verified for correction by the development team. It provides a contract for the *Defects fixed* (DF), which is required by the *Test execution* activity, thus creating the loop. It also provides the *Test defects log* (TDL), which it uses itself in further iterations. This activity is performed by the *Quality assurance lead*, *Module lead*, testing team and the development team.

The final activity of *TB-Enterprise testing process* is *UAT & closure* activity. On successful execution of all the tests, it receives VCDR, SSR, TC, TR artifacts at its required contracts. *User acceptance testing* is carried out as the final test of the software application. There are different *responsibilities* associated with this sub-activity. It is performed by user representatives, assisted by the testing team and approved by client and technical lead. Finally a closure of the testing process prepares the status summary report that details the test completion results.

For the *AlphaSystem* project, *TB-Enterprise* needs to follow the ISO standard 29119-2 as well. In order to comply with this process standard, it is developed as a specification process model. *ISO 29119-2 Test process* is a specification process model that is developed using *Process Specification metamodel* provided by CPMF framework. Once this model is developed, it can be stored in the process repository for reuse in other software development projects. *ISO 29119-2 Test process* combines the two layers of the standard; *Test management processes* and *Dynamic test processes*. *Test management processes* consists of three activities: *Test planning*, *Test monitoring & control* and *Test completion*. *Dynamic test processes* contains four activities, the *Test design & implementation*, *Test environment setup & maintenance*, *Test execution* and *Test incident reporting*. All the activities of *Dynamic test processes* are composed by a single parent activity, *Dynamic test*. The reasons for combining the two layers was to get a single process specification model for this standard, as shown in figure 6.4.

Test planning activity is responsible for the development of the test plan. Sub-activities are responsible for analyzing the context and organizing the plan development activity. Then risks are identified and the measures for treating them are devised. Further sub-activities are responsible for defining test strategy, determining scheduling & staffing and finally the development of *Test plan* (TP). Once TP is ready *Test monitoring & control* activity can be started. A dependency between them is highlighted through a binding between the provided contract of *Test planning* activity and required contract of *Test monitoring & control* activity for TP. *Test monitoring & control* is also a management level process, that is responsible for monitoring, control and reporting of the *Dynamic test* activity. This monitoring and control is carried out throughout the lifecycle of the *Dynamic test* activity. Once the *Dynamic test* activity is complete, it provides *Test results* (TR) and *Test summary report*(TSR) to the *Test completion* activity. *Test completion* activity archives the test artifacts, cleans up the test environment and reports the completion of the testing process. It provides the *Test summary report* for the future activities in the software development project.

Dynamic test activity of this standard is the activity responsible for designing, developing and executing tests for the software application. It is developed through four sub-activities: *Test design & implementation*, *Test environment setup & maintenance*, *Test execution* and *Test incident reporting*. *Test design & implementation* activity is responsible for the development of *Test case specifications* (TCS), *Test procedure specifications* (TPS) and *Traceability matrix* (TM). These artifacts are offered through the provided contract. Both *Test environment setup & maintenance* and *Test execution* activities show their dependencies to *Test design & implementation* activity for TCS, TPS, TM. *Test environment setup & maintenance* activity sets up the environment to carry out the tests. Once the environment is setup, a *Test environment readiness report* (TERR) is provided by this activity. This activity is also responsible for maintaining the test environment through multiple iterations.

Test execution activity depends on the *Test design & implementation* activity for TPS, TCS and TM and on *Test environment setup & maintenance* activity for *TERR*. Once the test environment is set up and the test cases are ready, this activity can execute the tests, compare test results and prepare the test execution log. This activity is generic and can be used for all types of tests being executed in the testing process. Successful completion of all tests, terminate it and eventually the *Dynamic test* activity. However, if the test fails, it provides the *Test execution log* (TEL) and *Test results*, which can be accessed by the *Test incident reporting* activity. *Test*

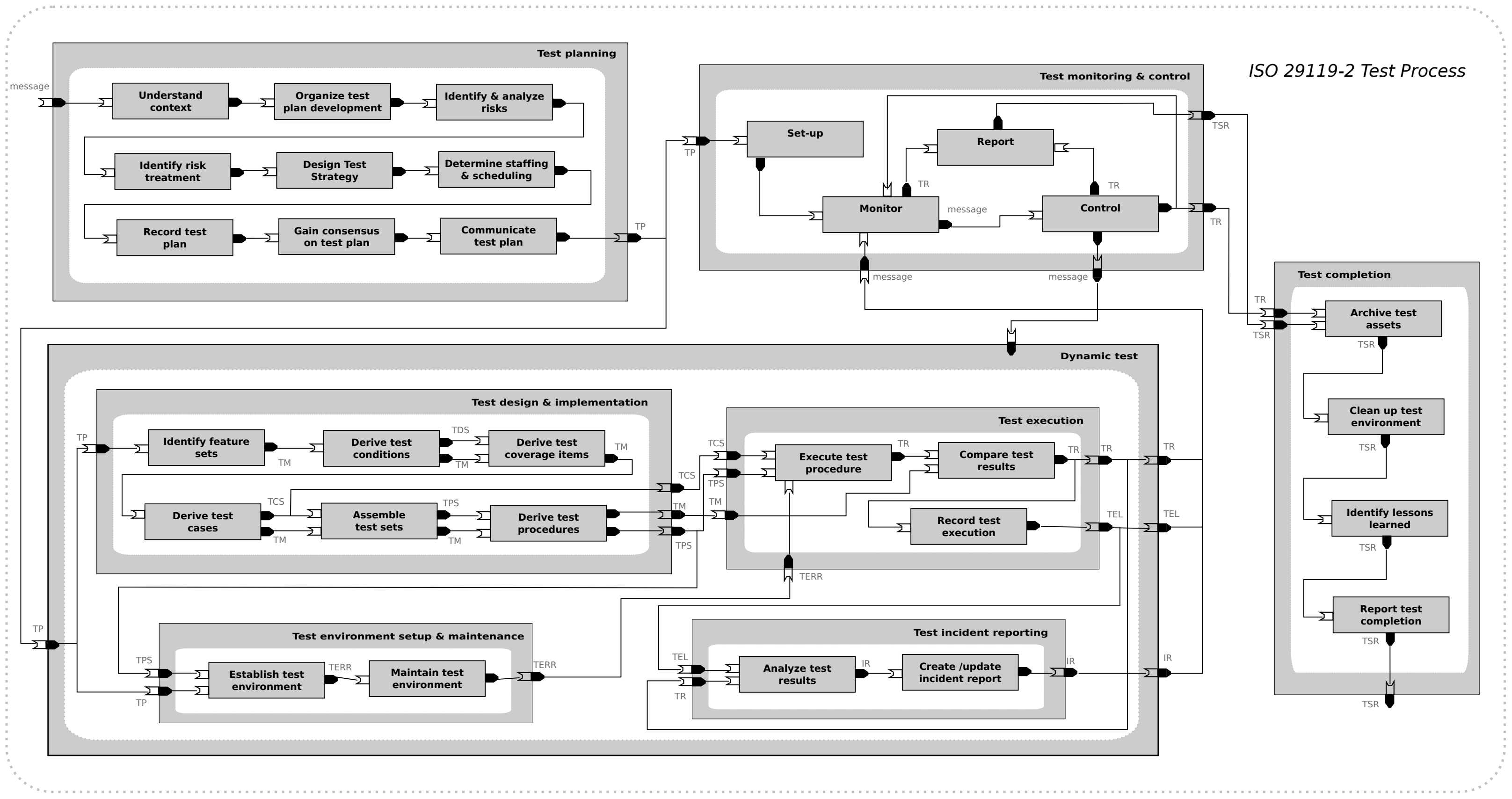


Figure 6.4 – ISO standard for testing process

incident reporting activity analyzes the test results and generates an *Incident report* for the *Test management processes.*

These two standards (organizational and ISO standard) for the testing process in software projects have the same objective, but have a different perspective on the process. ISO standard does not detail the specifics of different kinds of tests needed and their order. The organizational standard adds the debugging activities to the testing process as well. Compliance to both these standards for developing the testing process for *AlphaSystem*, requires a single *process implementation model* that should be refined from both these models. Process implementation model can use different names for the activities, but should keep the mapping towards the corresponding activity in either or both of these process specification models.

6.2.2 Design by Contract

Each activity defined in the process specification models presented in the previous section interacts with other activities through defined contracts. The inputs of the activities are specified through the required contracts and the outputs from the activities are specified through the provided contracts. For example *Analysis & planning* sub-activity of the *Test planning* activity in *TB-Enterprise testing process* has three required contracts: PMP, SDD and SRD and offers two provided contracts: TP and TS. Each of these contracts (whether required or provided) contains an artifact specification. All the work products specified in these two process models are artifact specifications and not the artifacts themselves. These artifact specifications describe the structure and function of the artifacts that need to be produced during the execution of these processes. PMP required contract of *Analysis & planning* activity presents an artifact specification of project management plan.

Bindings between the activity contracts ensure that the artifact required by an activity is the same as the artifact provided by the activity on which it is depending. Artifact specifications contained by both the contracts ensure the agreement over the artifact. This agreement can be further refined by the use of conditions. Preconditions and the post-conditions are defined for every activity (when needed) in these process models. Let us take the example of the *Integration testing* activity of *TB-Enterprise testing process.* The conditions associated with this activity are as follows:

Pre-conditions

— All functions in the build have successfully passed unit testing.

— The build is properly version controlled.

— Testing environment is in place for the test.

— Test cases include the cases for integration testing.

— All required integrated systems are available.

Post-conditions

— *Test result*(TR) report is developed/updated.

— Failed test have been added to the TR report with highlighted defects.

— Successful tests have been added in the TR report (for validation of bugs fixed in subsequent iterations).

6.2.3 Bi-layered implementation of processes

Once both the process specification models for the testing process are at hand, they can be refined for the development of the process implementation model. The process implementation model for the *AlphaSystem* testing process is a bi-layered model conforming to the *Process Implementation metamodel*, provided by the CPMF framework. The layers of the process model correspond to abstract and concrete levels. The abstract level of the process model contains the activity definitions and the bindings between them to define the flow of data. The concrete level of the process model contains the activity implementations, each of which implements an activity definition. Activity definitions at the abstract level are not hierarchical. The implementation of an activity definition at the concrete level decides whether the activity is implemented through multiple sub-activities or a single primitive activity.

Processes from the adopted standards (organizational standard and the ISO standard) for the *AlphaSystem* testing process are refined in a way that the abstract process of PImp model contains a set of all the abstract processes (containing activity definitions without their internal hierarchy). The internal hierarchy of each activity definition corresponds to a separate process, which is also present in the abstract level. Thus the abstract level of the process implementation model contains all the processes and sub-processes without any link of containment. Containment of a process in an activity definition of some other process is dependent on the implementation of this activity definition. Figure 6.5 presents the implementation process model for the testing process. However all other activities definitions except *Test Execution Def* are intentionally not depicted in this figure for reasons of brevity. *Test Execution Def* activity definition at the abstract level presents its external contracts, but does not give any detail about its implementation. *Test Execution Abstract Process* is also present at the abstract level. It was contained by the *Test execution* activity in specification model. But at the abstract level of implementation model, it is a separate process having no link with *Test execution Def*. However it is a possible candidate process that can be used for the implementation of this activity. *AlphaSystem Test Execution Activity* implements the *Test Execution Def* activity definition by providing a concrete process corresponding to the *Test Execution Abstract Process.*

Test Execution Def activity defined at the abstract level presents its (input and output) *artifact contracts*. Besides the *artifact contracts* it also presents its lifecycle contract (not depicted in the figure), that presents the state machine for this activity definition. Even though, CPMF framework allows a custom life cycle for each activity definition, it uses a default lifecycle with six activity states: *waiting*, *ready*, *active*,

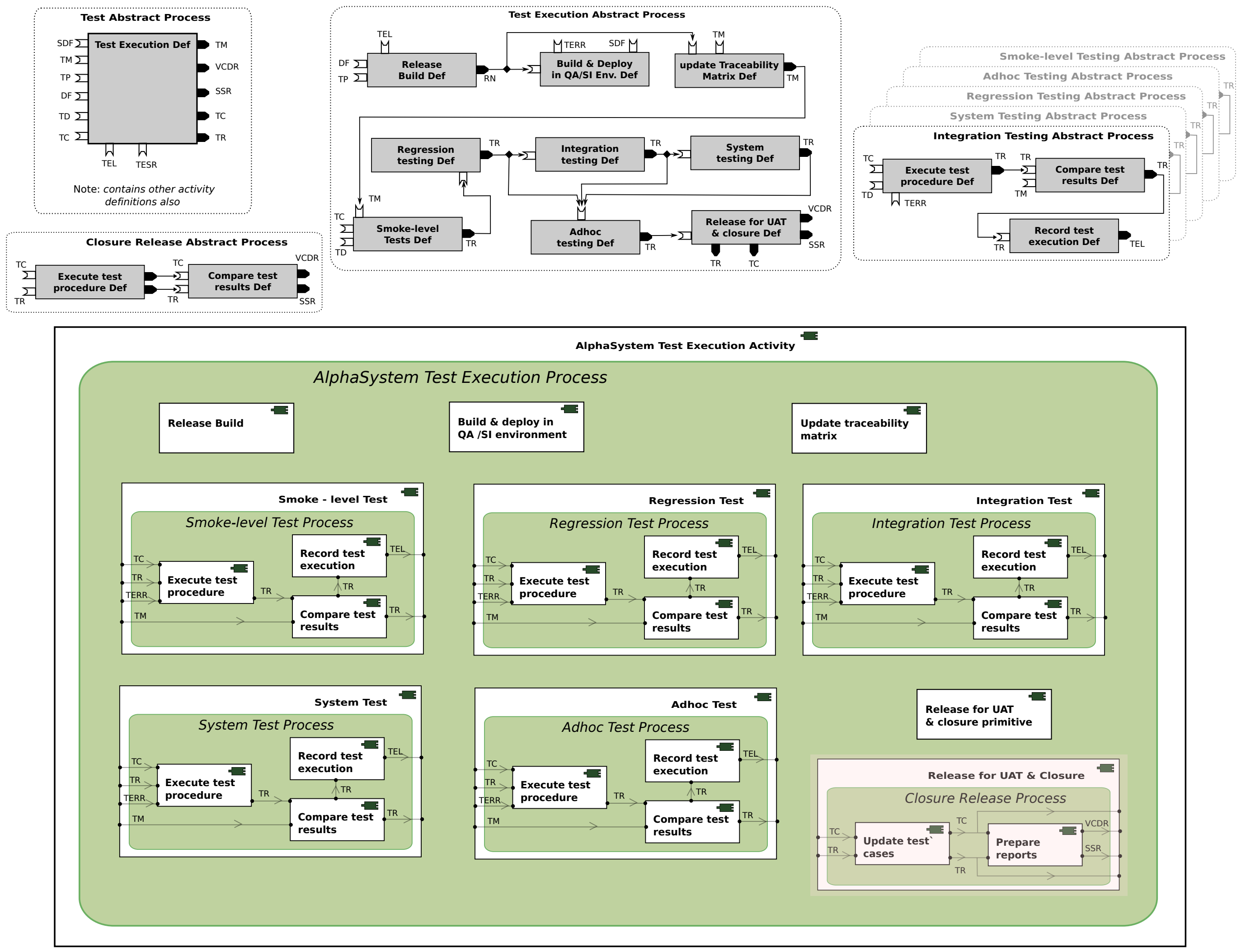

Test Abstract Process
Test Execution Def
SDF
TM
TP
DF
TD
TC
VCDR
SSR
TR
TEL
TESR
Note: contains other activity definitions also
Closure Release Abstract Process
Execute test procedure Def
Compare test results Def
Test Execution Abstract Process
Release Build Def
Build & Deploy in QA/SI Env. Def
update Traceability Matrix Def
Regression testing Def
Integration testing Def
System testing Def
Smoke-level Tests Def
Adhoc testing Def
Release for UAT & closure Def
RN
TERR
Smoke-level Testing Abstract Process
Adhoc Testing Abstract Process
Regression Testing Abstract Process
System Testing Abstract Process
Integration Testing Abstract Process
Record test execution Def
AlphaSystem Test Execution Activity
AlphaSystem Test Execution Process
Release Build
Build & deploy in QA /SI environment
Update traceability matrix
Smoke - level Test
Smoke-level Test Process
Regression Test
Regression Test Process
Integration Test
Integration Test Process
System Test
System Test Process
Adhoc Test
Adhoc Test Process
Execute test procedure
Record test execution
Compare test results
Release for UAT & closure primitive
Release for UAT & Closure
Closure Release Process
Update test` cases
Prepare reports

Figure 6.5 – Implementation model for test execution

paused, *terminated* and *completed.* The events in this state chart are *prepare*, *activate*, *terminate*, *complete*, *pause* and *resume.* The state machine defined in this activity life cycle contract, links these events to corresponding states for triggering a change in state.

Each artifact contract of the *Test Execution Def* presents the *artifact specification* for the artifact that it provides or requires. This artifact specification also provides a metamodel for the artifact. For example, the *Test Cases*(TC) artifact contract for this activity defines the artifact specification for test cases. This particular artifact specification requires that a test case should have a name, description and data requirements. The associated metamodel for the test case provides its structure consisting of multiple steps, where each step has its description and the expected results. TC artifact contract of *Test Execution Def* also presents a state machine for the artifact lifecycle.

AlphaSystem Test Execution Activity at the concrete level implements the *Test Execution Def* activity by providing the implementation details through the *AlphaSystem Test Execution Process.* As it implements the corresponding activity definition as a composite activity, opposite internal contracts are created for all its external contracts. Each contract of this activity implementation presents a set of events that map to the artifact specification at the abstract level. For example, the TC contract of this activity implementation presents a set of events that map of the test case specification at the abstract level. *AlphaSystem Test Execution Activity* has two TC contracts, one required and one provided. Each of the implementation level contracts, maps to the respective contracts at the abstract level. The set of events presented by the provided TC contract at the implementation level are the subset of events defined in the artifact state machine. However the set of events (actually event listeners) present at the required TC contract are the super-set of the events defined in the corresponding artifact state machine. Each pair of internal and external contracts, serves for the delegation of events across activity borders. The internal lifecycle contract of the *AlphaSystem Test Execution Activity* presents an event broker that is responsible for managing the interactions between the sub-activities.

6.2.4 Reusing process elements

The abstract level of the implementation model for *AlphaSystem Tesing process* contains the set of abstract processes that are used for activity implementations. Each abstract process for a specific kind of testing like integration testing, system testing, *etc.* follow the same hierarchical structure. The reuse of processes at the abstract level is demonstrated by reusing the *Integration Testing Abstract Process* for the development of other abstract processes for system testing like regression testing *etc.* Once a process is developed, it can be reused and tailored for a specific use to model other processes. This reuse of abstract process remains at the implementation phase of process development only, because tailoring abstract level processes in instantiation phase is currently not handled by the associated tool implementation.

As an abstract process is reused within a process model, similarly the concrete level activity implementations are also reused for concrete processes. *AlphaSystem Test Execution* activity has five activities for different kinds of tests. As the contracts of the corresponding activity definitions for these activities are the same, a single implementation is reused for implementing all these activities. This should not be confused with activity sharing, where one activity implementation is shared amongst different processes. In case of the sub-activities of *AlphaSystem Text Execution*, they are all different implementations having different properties, but their development is based on the reuse of a single implementation.

Release for UAT & Closure activity in this model is implemented as a primitive activity. Let us assume that an already developed activity definition & implementation for this activity is found in the process repository. By searching for the associated tags and contracts, these process elements can be retrieved from the process repository. Once this second implementation is retrieved, it can be added to the current process model. Implementation level process model allows multiple activity implementations for an activity definition. In order to add this implementation to the model, a copy of the abstract level process for its contained activity architecture is placed in the abstract level of the current process model and a copy of the activity implementation is added to the concrete level.

6.2.5 Process refinement

This case study deals with two specification process models described in the previous sections. These process specification models are refined as a single process implementation model. Process specification metamodel for the *AlphaSystem* test process was defined in a single level without any implementation details, other than the process hierarchy. This process hierarchy is only a part of the complete process implementation. When a process model is refined to the PImp model, implementation details are injected into this process model. The structure of the process model changes from a single layer to a bi-layered architecture. Once the two layers are formed, the abstract level of the process contains the structural information for the activity regarding its contracts and the data-flow architecture of the processes. The concrete level of the process offers the concrete activity implementations. The injection of implementation details accounts for the addition of activity life cycles, artifact life cycles, artifact specification metamodels, *etc.* at the abstract level. Implementation details that are injected in the concrete level of the process model are the definition of events to specify the control-flow, properties of the activities and definition of milestones *etc.* that are specific to the particular project under development. These implementation level details are injected into the *AlphaSystem Test Execution* activity, once it is refined as an implementation model from the specification models.

This process model gets further refined into the process instantiation model, where the abstract level of the model remains more or less the same as that of the process implementation model, as depicted in figure 6.6. However, further instantiation level details are injected into the concrete level of this process model. The reason we say that the abstract level remains more or less the same is that no structural change occurs at the abstract level, other than hiding the activity state machines of the required contracts and only showing the list of events. The validations carried out at the implementation level between provided and required artifact contracts of interacting activities are not repeated in this phase. Because we do not allow adaptations of the abstract level of the instantiation process model for now, these validations become redundant. The concrete level activity implementations of PImp model are refined into instantiation activities. These instantiation activities require additional properties for staffing, scheduling, handling artifact in repositories, message choreographies *etc.* Staffing details are the assignment of roles to actors. For example the *Execute Test Procedure* has *Tester* as an associated role with *Responsible* as its assigned responsibility. This means that tester is responsible for carrying out this activity. The *Tester* role is played by an actor named Marianne. The profile of an *actor* gives addition details like email, department, organization, *etc.* for realizing the communications. Scheduling details for this activity give the start date & time and the end date & time.

An artifact is created for each artifact specification defined at the abstract level. In case the artifact is a hard copy (not digital), a dummy object is created with its associated properties to manage the physical distribution/transfer of the artifact between the stakeholders. In this particular case study, most of the artifacts passed between the activities were digital copies, thus a repository url address was assigned to each artifact. This repository address helps in retrieving the artifact from the repository by the activity that requires it. The *Test Environment Readiness Report*(TERR) required by the *Build & Deploy in QA/SI environment* activity is the only hard copy artifact. For this artifact, a dummy artifact is created that shows the current possessor of this artifact. It also keeps track of the state of the artifact, as per the artifact state machine defined for it at the abstract level.

6.2.6 Execution of scenario processes

Process instantiation models are executable. The process interpreter, developed as a tool support for this research project, is responsible for executing the processes. A project management dashboard is a web interface provided alongside the interpreter that serves for monitoring and controlling the execution of the software development process models. Each actor plays certain roles in a project. The authentication module of the project management dashboard allows access to the concerned actors only. The transitions of artifact and activity states in a software project are associated with specified roles for each activity. Thus an actor can only trigger the transitions that are authorized to the role(s) that it is playing. Thus when an actor logs into the project management dashboard, he/she can only view the activities associated to him/her. This actor can only perform the actions that are authorized to him/her in the project management dashboard.

Marianne is a tester in *TB-Enterprise*. She physically receives the *Test Environment Readiness Report* (TERR) on a working day. When she logs in to her project

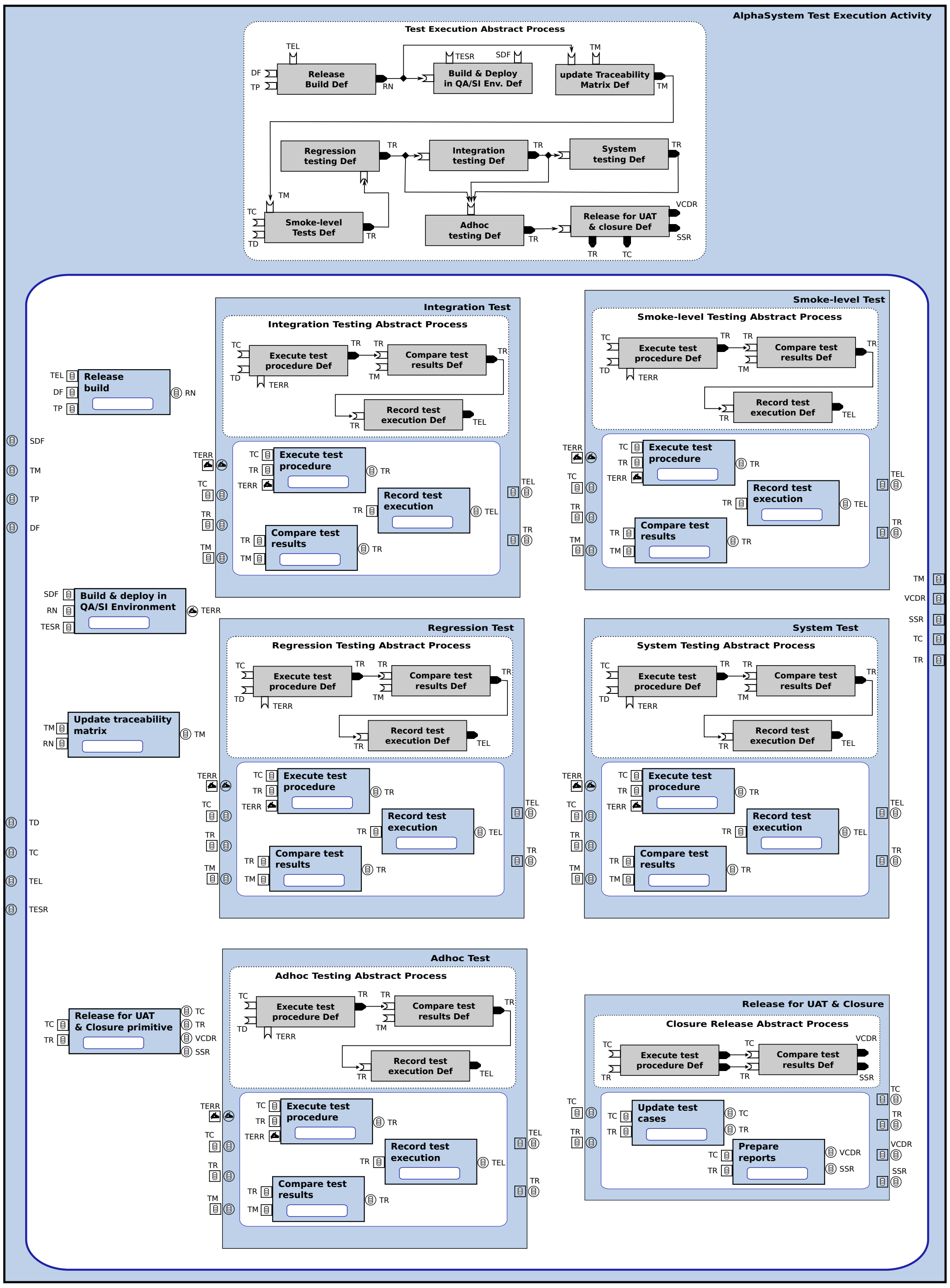

AlphaSystem Test Execution Activity
Test Execution Abstract Process
Release Build Def
Build & Deploy in QA/SI Env. Def
update Traceability Matrix Def
Regression testing Def
Integration testing Def
System testing Def
Smoke-level Tests Def
Adhoc testing Def
Release for UAT & closure Def
Integration Test
Integration Testing Abstract Process
Smoke-level Test
Smoke-level Testing Abstract Process
Regression Test
Regression Testing Abstract Process
System Test
System Testing Abstract Process
Adhoc Test
Adhoc Testing Abstract Process
Release for UAT & Closure
Closure Release Abstract Process
Execute test procedure Def
Compare test results Def
Record test execution Def
Execute test procedure
Record test execution
Compare test results
Release build
Build & deploy in QA/SI Environment
Update traceability matrix
Release for UAT & Closure primitive
Update test cases
Prepare reports
TEL
DF
TP
RN
SDF
TM
TD
TC
TESR
TERR
TR
VCDR
SSR

Figure 6.6 – Instance model for test execution

management dashboard she can enter into any of the projects that she is working on. For this case study she enters into the *AlphaSystem* project. She is responsible for carrying out the smoke tests for this project. The first build is already released and deployed in the quality assurance and system integration environment. The traceability matrix has been updated. When she logs into her dashboard for this project, she is notified that *Test Cases*(TC) and *Traceability Matrix*(TM) are available to her and the *Execute Test Procedure* sub-activity of the *Integration Test* activity is in 'ready' state. She can see from the information panel that she has to complete the tests on the same day. Links are provided on the dashboard so that she can download TC and TM.

The pre-conditions of the *Integration Test* activity ensure that the inputs to the activity are valid. The first test case is about testing the authentication module of the *AlphaSystem*. The steps defined in the test case along with their expected results are as follows:

— **Step 1:** Invoke application from desktop icon - (*Expected result*: Login screen is displayed)

— **Step 2:** Login with **Username** and **Password** - (*Expected result*: Main menu screen is displayed)

— **Step 3:** Logout from application - (*Expected result*: Login screen is displayed)

— **Step 4:** Login with **Incorrect Username** and **Password** - (*Expected result*: Login error message is displayed)

— **Step 5:** Login with **Username** and **Incorrect Password** - (*Expected result*: Login error message is displayed)

She starts the *Execute Test Procedure* activity by changing the state of the activity from ready to active. This automatically changes the state of the parent activity *Integration Test* from ready to active. She performs these steps in parallel with the second sub-activity *Compare Test Results* and notes the success/failure of the steps in the *TestResult*(TR) document. She can upload the TR document through dashboard as many times as she wants. Each time she uploads the TR document, a new version is created in the artifact repository. There are other test cases in the TC document as well, which she has to perform to complete this activity. Meanwhile she receives a phone call by the testing lead to pause the activity and do some other activities. She pauses the *Integration Test* activity, which automatically pauses all the running sub-activities. She resumes these activities once she gets the message from the testing lead. Once all the tests are executed and all the test results are compared and updated in the TR document. She uploads the TR document through the dashboard and explicitly specifies that it is the final version of this artifact. This triggers a transition of state for this artifact to complete and an event is triggered for the activities that required this document. Then she changes the state of these activities as complete and starts the *Record Test Execution* activity that she has to complete within the same working day in order to follow the project management plan.

Let us assume different exceptional situations during this process. 1) If the TC document to be received by *Integration Test* activity did not conform to the TC metamodel. The pre-condition of this activity would not allow this activity to receive this document. 2) The *AlphaSystem Test Execution* activity is paused or terminated during the execution of this activity. The *Integration Test* activity would be paused or terminated automatically. This is handled by the lifecycle contracts of each activity and the propagation of events from child to parent or parent to child (depending upon the event). 3) The failure of test cases in this activity is not considered as an exceptional situation. So in this case, the negative results are updated in the TR and the execution of the activity terminates normally.

6.2.7 Runtime adaptation

Tailoring a process to develop implementation process model or the instantiation process model is possible by loading the process in the process editor and carry out the necessary changes. Process tailoring may be needed for improving the current process model or for customizing it. Process models are tailored by customization when they are retrieved from the process repository to build a new model. Apart from process tailoring, CPMF framework also allows for runtime adaptations of the process. This runtime process adaptation allows to change processes during the execution of the process model. The adaptation allowed by CPMF framework has certain limitations: 1) It only offers a controlled adaptation. This means that different process variations are already at hand and the process engineer can choose to interchange them during execution (with or without state transfer). 2) Process variations account for the concrete level of the process only *i.e.* abstract processes can not be adapted during execution. For the adaptions that are not supported by CPMF framework yet, the running process has to be stopped. And the implementation process model needs to be tailored, then refined into the instance process model for execution.

AlphaSystem Test Execution Activity is developed at the implementation phase of process development lifecycle in this case study. This activity implementation was developed using the concrete process corresponding to the *Test Execution Abstract Process.* This concrete process is *AlphaSystem Test Execution process* that contains the activity implementations of all the activity definitions present in the *Test Execution Abstract Process.* One of these activity definitions is *Release for UAT & Closure Def.* Two activity implementations were developed for this activity definition at the concrete level, as depicted in figure 6.6. One of them is a primitive activity implementation and the other is a composite activity implementation. The composite activity implementation is implemented through the *Closure Release Process.* In order to implement this activity with this process, its corresponding abstract process, *Closure Release Abstract Process*, is added to the abstract level of the process model. This provides a complete process model, that can use any of these activity implementations, where the abstract processes for both these implementations are present at the abstract level. When multiple activity implementations are developed for a single activity definition in a process model, one of them is selected to be the 'active' implementation. In this case, we have chosen the composite activity implementation to be the active implementation for the process model.

When the process model is refined to the instantiation process model, the abstract level of the process remains almost the same, but is included in the parent activity implementation, as shown in figure 6.6. Runtime process adaptation does not allow any adaptations to the abstract level of the instance process model. Activity implementations of the *Release for UAT & Closure Def* activity definition in implementation model are refined as instantiation activities at the concrete level of instance process model. When this process model is executed, the interpreter loads the 'active' instantiation activity i.e. the *Release for UAT & Closure* instantiation activity. During the execution of the process model, the process engineer decides to adapt the process model in the way that he wants to replace *Release for UAT & Closure* instantiation activity with *Release for UAT & Closure Primitive.* As both these instantiation activities implemented the same activity definition, thus the contracts of both these instantiation activities allow to replace one with the other. The state of *Release for UAT & Closure* might have been updated during the execution of the process model. In order to replace them during the execution of the process model, the state of *Release for UAT & Closure* needs to be transferred to *Release for UAT & Closure Primite* instantiation activity. CPMF framework provides two options to the process engineer to transfer the state: 1) through the use of a constructor, where the mapping between the properties of the instantiation activities is already developed in the constructor. 2) through manual mapping, where the properties of the instantiation activities are manually mapped, so that the state of one instantiation activity can be transferred to the other. In this specific case study, as we had only two variations of implementations, so the transfer of state is carried out through the use of constructor. However, this can also be performed through manual mapping, which can be carried out through the project management dashboard. Project management dashboard offers a html form showing the current state of the activity and demands the value of the new state. It is important to understand that process adaptations are not carried out by the responsible roles of the activities. They can only be carried out by the *process owner*, which is a special responsibility associated with composite activities (that contain the process).

6.3 Findings & Discussion

This case study involves the implementation of the testing process of a software development project, *AlphaSystem*, in a fictitious company named *TB-Enterprise.* CPMF framework proposes to model software development processes using a dedicated metamodel for each phase of software process development life cycle. A dedicated metamodel for a particular phase of process development life cycle allows to define a boundary around the concepts relating to this specific phase. Consequently, the process model in a particular phase does not get polluted with the irrelevant concepts in that phase. This produces a lightweight process model in the initial phase of process development that gets refined over time with each passing phase. Additional details are added to the process model in the appropriate phase of process development. *TB-Enterprise* uses this ideology to develop the *AlphaSystem testing process* through a set of models, each corresponding to a specific phase of process development life cycle.

TB-Enterprise had already developed its own organizational standard for software testing process for a previous project. This standard was developed using the CPMF approach and thus the corresponding process model was stored in the process repository at the time of its development. For *AlphaSystem* project, the client has demanded a compliance to ISO Standard 29119-2, which resulted in a scenario where *TB-Enterprise* has to comply with both the standards. It develops a process specification model for the ISO standard, which is also stored in the process repository for possible future reuse. Both these process specification models are refined into a process implementation model. Process implementation models are project specific, where specific implementation details for the process are injected into the process model. This process model is further refined into an instance process model that allows its execution by the process interpreter.

This case study focuses on explaining the key propositions made in the thesis. CPMF approach for dealing with specific scenarios was demonstrated, which are difficult to handle otherwise. Different process modeling approaches, discussed as the state of the art in this thesis, offer little or no support for compliance with process standards. CPMF approach not only offers the possibility to comply with a process standard, it takes a step ahead by offering compliance to multiple standards. This is demonstrated in this case study through the use of two standards (*i.e.* an organizational standard and a ISO standard), where *AlphaSystem Testing Process* complies to both of them.

One of the main focuses of CPMF approach is to foster decoupling of activities in process models. This is achieved through the use of events based flow at the concrete level of executable process models and the completeness of the definition for each individual activity. A single activity without its surrounding processes is not able to give a meaningful information in other process modeling approaches. On the contrary, a single activity in an executable CPMF process model contains the details of the associated work products through artifact specifications, artifact state machines, artifact metamodels, the details of activity life cycle, associated pre-conditions and post-conditions, etc. This completeness of the definition of an activity is achieved over time through a series of refinements. This is also demonstrated by the implementation of the processes in this case study. Apart from the completeness of activities, this case study demonstrates the design by contract approach followed by CPMF to foster correctness of interactions and modularity.

This case study also demonstrates the way actors interact with executing process models through the project management dashboard. How their access rights to certain activities and certain actions related to a specific activity are managed by the dashboard. Current process modeling approaches do not focus on the runtime adaptations of the executing processes. CPMF also takes a step forward in this direction to make software processes more dynamic. CPMF offers a 'controlled' runtime adaptation for executing processes in a way that a process can be adapted to different variations already developed in the implementation phase. A scenario of process adaptation is also covered in this case study that demonstrates the approaches for runtime adaptations.

Chapter 7

Pattern Support in CPMF

Contents

Abstract *- This chapter presents the implementation of workflow patterns using CPMF Framework. It also compares the results with other well-known approaches for software and business process modeling. We have focused on three types of patterns for workflow: data-flow, control-flow and resource-flow. We conclude this chapter with a discussion that presents the overall evaluation of the CPMF approach based on the implementations of the workflow patterns that it supports.*

7.1 Workflow Patterns

Workflow patterns are developed under the workflow pattern initiative by Eindhoven University of Technology and Queensland University of Technology. This initiative started in 1999, which initially presented workflow control-flow patterns [van der Aalst 03a]. A total of 20 patterns were presented to define the control-flow perspectives that should be offered by a complete process modeling language or a workflow language. Later on, another 23 control-flow patterns were added to the original control-flow patterns, thus a total of 43 patterns [Russell 06a]. The semantics of these control-flow patterns is defined through Coloured Petri Nets. Workflow pattern initiative also presented workflow patterns for data [Russell 05a], resource [Russell 05b] and exception handling [Russell 06b] perspectives. The purpose of presenting workflow patterns for a variety of perspectives is to assess the relative strengths and weaknesses of different process modeling approaches.

CPMF approach is assessed against all the data, control-flow and resource perspectives offered by the workflow patterns. As the proposed process modeling approach does not provide support for exception handling for the moment, we did not present an evaluation of this perspective. Workflow pattern initiative provides the evaluated rating for well-known process modeling languages and tools. This gives us an opportunity to place the evaluations of CPMF approach alongside other approaches for the purpose of comparisons. The details of each pattern are not included in this chapter for reasons of brevity. They can be accessed on the workflow pattern initiative website[1].

7.2 Workflow Data Patterns

The flow of data from one activity to another in an executing process, enables it to produce meaningful results. Different approaches focus on different aspects to model the processes. Some of them keep the flow of data as their primary focus and the process model is built around the concept of data-flow between the activities. Others may choose to be guided by the control-flow for modeling processes. CPMF framework uses a bi-layered approach, where the data flow is specified at the abstract level of the process model. The concrete level of the process model handles the control-flow of the processes using an event management system. Even though data flow is specified at the abstract level, it is realized through the event management system during execution. Artifact events at the concrete level map to their respective artifact specifications. The flow of an event between two activities realizes the actual transfer of data between them.

The data to be transferred between the activities can be of different forms. CPMF supports the flow of data in two forms: *artifacts* and *messages*. Both forms of data are passed using their respective contracts. Artifacts in CPMF are structured documents that are considered as models. Each artifact conforms to its metamodel. The flow of data between two activities, specified at the abstract level, uses the corresponding artifact specifications. Apart from presenting the metamodel for the artifact, these artifact specifications also provide the artifact state-machine for defining its life cycle. *Messages* are also passed between the executing activities. They can pass data in the form of emails & text messages and also the data types like string, integer, float, boolean, date, time, *etc.*

Workflow data patterns[2] are the patterns defined by the Workflow pattern initiative, that concern the flow of data in a process[Russell 05a]. Table 7.1 summarizes the evaluation of CPMF implementation for the supported data patterns and compares it with other process modeling approaches. Evaluation results of other process modeling frameworks/tools in this table are taken from the Phd thesis of Nick Russell [Russell 07]. Details for CPMF Implementations of these data patterns can be consulted in appendix 1.1. First eight *Workflow Data Patterns*(WDP) concern the

1. http://www.workflowpatterns.com/
2. http://www.workflowpatterns.com/patterns/data/

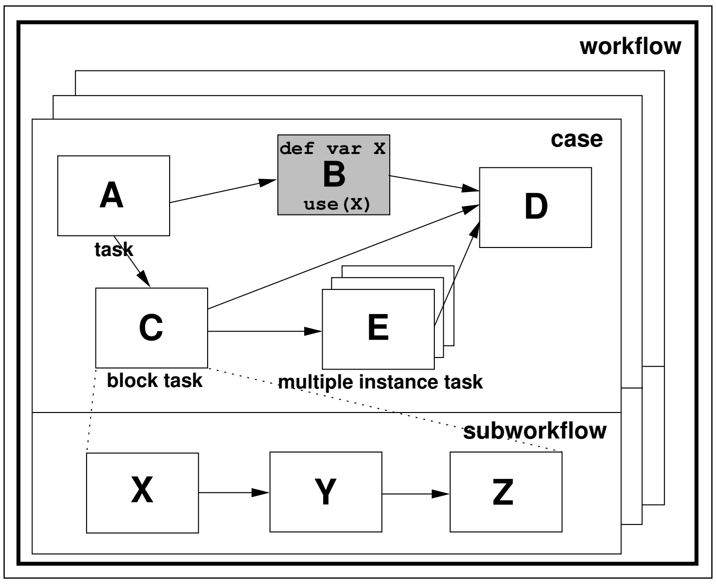


Figure 7.1 – Task level data visibility [Russell 05a]

visibility of data elements within a process model. For example WDP-1 presents a pattern where a data element's visibility is restricted only to the context of individual execution instance of a task. This pattern is presented by the Workflow Patterns Initiative [Russell 05a], as:

Pattern 1 (Task Data)
Description: Data elements can be defined by tasks which are accessible only within the context of individual execution instances of that task.
Example The working trajectory variable is only used within the Calculate Flight Path task.
Motivation To provide data support for local operations at task level. Typically these data elements will be used to provide working storage during task execution for control data or intermediate results in the manipulation of production data. Figure 7.1 illustrates the declaration of a task data element (variable X in task B) and the scope in which it can be utilised (shown by the shaded region and the use() function). Note that it has a distinct existence (and potential value) for each instance of task B (i.e. in this example it is instantiated once for each workflow case since task B only runs once within each workflow).

CPMF framework encapsulates the data within the activity such that it can be shared with its context, only through its specified contracts. The visibility or scope of a data element in an activity depends upon the specification of artifact contracts and message contracts at the abstract level of the process model. WDP-1 to 3 concern different levels of scope for the elements *i.e.* within the context, to the contained

Pattern	WebSphere	FlOWer	COSA	XPDL	BPEL	BPMN	UML	CPMF
WDP-1 (Task Data)	◓	◓	●	○	◓	●	◓	●
WDP-2 (Block Data)	●	●	●	●	○	●	●	●
WDP-3 (Scope Data)	○	◓	○	○	●	○	○	●
WDP-4 (Multiple Instance Data)	●	●	●	●	○	◓	●	●
WDP-5 (Case Data)	●	●	●	●	●	●	○	●
WDP-6 (Folder Data)	○	○	●	○	○	○	○	◓
WDP-7 (Workflow Data)	●	○	◓	◓	○	○	●	●
WDP-8 (Environment Data)	◓	●	●	○	●	○	○	●
WDP-9 (Data Interaction - Task to Task)	●	●	●	●	●	●	●	●
WDP-10 (Data Interaction - Block Task to Sub-Workflow Decomposition)	●	◓	◓	●	○	◓	●	●
WDP-11 (Data Interaction - Sub-Workflow Decomposition to Block Task)	●	◓	◓	●	○	◓	●	●
WDP-12 (Data Interaction - to Multiple Instance Task)	○	●	○	○	○	○	●	●
WDP-13 (Data Interaction - from Multiple Instance Task)	○	●	○	○	○	○	●	●
WDP-14 (Data Interaction - Case to Case)	◓	◓	●	◓	◓	○	○	◓
WDP-15 (Data Interaction - Task to Environment - Push-Oriented)	◓	●	●	●	●	●	○	◓
WDP-16 (Data Interaction - Environment to Task - Pull-Oriented)	◓	●	●	●	●	●	○	◓
WDP-17 (Data Interaction - Environment to Task - Push-Oriented)	◓	◓	●	○	◓	●	○	◓
WDP-18 (Data Interaction - Task to Environment - Pull-Oriented)	◓	◓	●	○	◓	●	○	◓
WDP-19 (Data Interaction - Case to Environment - Push-Oriented)	○	●	○	○	○	○	○	◓
WDP-20 (Data Interaction - Environment to Case - Pull-Oriented)	○	●	○	○	○	○	○	◓
WDP-21 (Data Interaction - Environment to Case - Push-Oriented)	◓	●	●	○	○	○	○	◓
WDP-22 (Data Interaction - Case to Environment - Pull-Oriented)	○	●	●	○	○	○	○	◓

Table 7.1 – Workflow Data Patterns WDP-1 to WDP-22

activities and to the subset of contained activities. CPMF framework supports all three kinds of scope for a data element.

WDP-4 defines the scope of a data element restricted to the executing instance of an activity that can have multiple instances. It is also supported by CPMF approach, which allows multiple instances of an activity. Each activity instance has its own working copy of the artifacts. WDP-5 defines a *case* scope for the data element. A case is a particular instance of a process instead of an activity. In CPMF, a case corresponds to an instance of a composite instantiation activity, which follows the same contractual interaction paradigm. WDP-6 defines a scope for the data element such that it can be assessed by multiple cases on a selective basis. CPMF allows to bind data with multiple composite activities. The use of pre-conditions can be exploited for selective accessibility of the data elements, providing a partial support for the implementation of WDP-6. A data element is accessible by any activity in the process model that defines a corresponding required contract for it, thus supporting WDP-7. WDP-8 requires the approach to be able to support data elements that are present in the execution environment of the process model *e.g.* from other applications. Roles are associated to CPMF activities, which may be played by actors (human resource) or tools. These tools allow the activities to access or provide information to the environment. Interactions with the executing environment are not contractual.

The data patterns from WDP-9 to 14 concern the data interactions within a process. WDP-9 is supported by CPMF, because any activity is able to interact with another activity within the same process instance. WDP-10 requires a composite activity to be able to interact with its contained activities. This is possible through the use of internal contracts of the composite activities. Internal contracts of an activity can either be required or provided. These interactions are possible in both ways, child to parent and parent to child, hence supporting WDP-11. WDP-12 and WDP-13 require an activity to interact with multiple instances of another activity. CPMF framework allows multiple instances of an activity at runtime. Interactions to/from them are also possible. These interactions can be selective through the use of conditions associated with the activities. This allows CPMF to implement both these patterns. To implement WDP-14, two instances of a single composite activity should be able to interact. To implement such a pattern, the provided contract of the composite activity needs to be 'bound' to the corresponding required contract of the same activity. Only partial support for such a pattern is available, as CPMF does not support it directly. Pre-conditions on the activity instances and the locking mechanisms of the repository need to be exploited to achieve this behavior.

The data patterns from WDP-15 to WDP-26 are related to the data interactions of the process elements with external environment. All the activities in CPMF framework define associated roles for performing these activities. Multiple roles can be associated with an activity, each having a different responsibility. These roles can be played by actors (human resource) or by tools. Tools associated with the activities are responsible for interactions with the environment. These interactions are not contractual, as in the case of inter-activity interactions. Extra programmatic extensions may be required to connect the tools with activity implementations (specifically the automatic activities). We rate these patterns to be partially supported by CPMF, as per the defined evaluation criteria of these patterns [Russell 05a].

Table 7.2 summarizes the evaluated ratings for the rest of the workflow data patterns. WDP-27 to WDP-33 are the data transfer patterns, that describe the mechanism by which data elements are passed from one activity to another. WDP-27 & 28 describe the transfer of data elements from one activity to another as 'transfer by value'. The transfer of artifacts from one activity to another in CPMF takes place through a common artifact repository. Thus, CPMF activities share a common address space to provide or require an artifact. CPMF allows 'transfer by reference' only, hence supporting WDP-30 & 31 and not providing any support for WDP-27 & 28. WDP-29 is supported by CPMF, as each activity keeps a local working copy of the artifact that it accesses from the artifact repository. WDP-32 requires the ability to perform a transforming function over the input data element just before its is passed to the activity. CPMF does not allow any 'action' to be performed outside an activity. Thus the transforming function just before the input needs to be developed as a separate activity, which is not the same intention, presented through this pattern. So WDP-32 is not supported. However,in WDP-33 the transformation function is performed just before it is passed out of the activity. This pattern is supported, as the transformation function is carried out inside the activity in this case.

The remaining workflow data patterns concern the routing of data from one activity to another. WDP-34 & 35 are supported by CPMF framework as preconditions on the activities can be based both on the existence of the data elements to their value. Every artifact in CPMF is considered as a model, whether it be a graphical model or a textual document. Furthermore, its life cycle and specifications defined at the abstract level make it a structured artifact. Pre-conditions associated with the activities can access the properties of these artifacts. Similarly, the post-conditions can also be based on the existence or the value of the data elements, thus providing direct support for WDP-36 & 37. WDP-38 demands an activity to be triggered by an external event. The contracts of activities at the concrete level of CPMF process model contain events. The provided contracts of the activities are responsible for triggering external events and the required contracts are responsible for listening to them. This data pattern is directly supported by CPMF. Data based task trigger required by WDP-39 is partially supported by CPMF, as no monitoring routines based on the value of data elements are available to the activities. However, the events notifying the change of state for the artifacts are listened by the activities, thus giving the possibility to trigger an activity. The last data pattern, WDP-40 that requires the data based routing is also partially supported by CPMF. Logical connectors are not explicitly specified in CPMF process models, their logic needs to be implemented in the contracts. Routing based on the value of the data elements can be achieved through these contracts along with pre-conditions and the event broker for the process.

Pattern	WebSphere	FlOWer	COSA	XPDL	BPEL	BPMN	UML	CPMF
WDP-23 (Data Interaction - Workflow to Environment - Push-Oriented)	◓	○	○	○	○	○	○	◓
WDP-24 (Data Interaction - Environment to Workflow - Pull-Oriented)	○	○	○	○	○	○	○	◓
WDP-25 (Data Interaction - Environment to Workflow - Push-Oriented)	◓	○	○	○	○	○	○	◓
WDP-26 (Data Interaction - Workflow to Environment - Pull-Oriented)	●	○	●	○	○	○	○	◓
WDP-27 (Data Transfer by Value - Incoming)	●	○	◓	◓	●	●	○	○
WDP-28 (Data Transfer by Value - Outgoing)	●	○	◓	◓	●	●	○	○
WDP-29 (Data Transfer - Copy In/Copy Out)	○	◓	○	◓	○	◓	○	●
WDP-30 (Data Transfer by Reference - Unlocked)	○	●	●	●	●	○	○	●
WDP-31 (Data Transfer by Reference - With Lock)	○	◓	○	○	◓	●	●	●
WDP-32 (Data Transformation - Input)	○	◓	○	○	○	◓	●	○
WDP-33 (Data Transformation - Output)	○	◓	○	○	○	◓	●	●
WDP-34 (Task Precondition - Data Existence)	○	●	●	○	◓	●	●	●
WDP-35 (Task Precondition - Data Value)	○	●	●	●	●	○	●	●
WDP-36 (Task Postcondition - Data Existence)	●	●	○	○	○	●	●	●
WDP-37 (Task Postcondition - Data Value)	●	●	○	○	○	○	●	●
WDP-38 (Event-based Task Trigger)	◓	●	●	○	●	●	●	●
WDP-39 (Data-based Task Trigger)	○	●	●	○	◓	●	○	◓
WDP-40 (Data-based Routing)	●	◓	●	●	●	●	●	◓

Table 7.2 – Workflow Data Patterns WDP-23 to WDP-40

7.3 Workflow Control-flow Patterns

Workflow Control-flow Patterns (WCP) are the patterns defined to describe the flow of control within the activities of a process model. This flow of control is based on different dependencies between the activities to transfer the execution control. These patterns initially cover the basic control flow logic like series, parallel execution, AND, OR and XOR like constructs. Later on, patterns related to iterations, cancellation, termination and multiple instances *etc.* are discussed. Originally, 20 control-flow patterns were defined to describe different scenarios [van der Aalst 03a]. After that, 23 additional patterns were added to make the set of control-flow patterns more comprehensive and complete [Russell 06a]. Some of the control flow patterns are said to be very specific to YAWL language, for comparing the process modeling languages [Börger 12]. However, we are going to implement all these control-flow patterns to get a comprehensive picture.

CPMF framework specifies the data-flow between the activities at the abstract level of its bi-layered process model. The control-flow of the activities is defined in the concrete level using events. The focus of CPMF methodology is not on describing the control flow of the activities in an easy manner. Rather, it focuses on more abstract concepts like separation of concerns, contractual interactions, process refinements etc. For this reason, the logical connectors are not explicitly specified outside the activities; they are encoded within the contract of CPMF activities. This may make it hard to understand the flow of control in a process model, visually. However, we believe that multiple views can be extracted from a complete definition of a process model that serve for a better understanding of a process model, for a particular point of view. For the moment, the implementation tool provided along with CPMF framework does not offer any of these views. However, such a view can be developed to extract some particular information in a particular way, from a given process model.

Table 7.3 summarizes the evaluations of the implementation of original workflow control-flow patterns from WCP-1 to WCP-20. It also compares the evaluated ratings against other process modeling approaches, where their rating are already presented [Russell 07]. WCP-1 to WCP-8 and WCP-16 define the basic control-flow patterns involving sequence & parallel controls and the logical connectors like AND, XOR and OR with merge and split kinds. All these basic logical connectors are not explicitly specified outside the activities in the model, however CPMF framework offers a direct support for implementing these patterns inside the contracts. The split patterns are encoded inside the provided contracts and the merge patterns are encoded inside the required contracts of an activity. Events are propagated within a context using broadcast or unicast. For each process, the container activity offers an event broker that manages all interactions with the contained process. Activities can listen to the events based on their types, associated tags or the associated sender activity (where it listens to all the instances of that sender). For every event received by an activity, it is logged by the event broker. This information helps to manage the control flow within a process for XOR-type constructs, where control flow should be passed to one execution branch only. The contracts of an activity have a blocking mechanism,

Pattern	WebSphere	FlOWer	COSA	BPEL	BPMN	UML	EPCs	CPMF
WCP-1 (Sequence)	●	●	●	●	●	●	●	●
WCP-2 (Parallel Split)	●	●	●	●	●	●	●	●
WCP-3 (Synchronization)	○	●	●	●	●	●	●	●
WCP-4 (Exclusive Choice)	●	●	●	●	●	●	●	●
WCP-5 (Simple Merge)	●	●	●	●	●	●	●	●
WCP-6 (Multi-Choice)	●	●	●	●	●	●	●	●
WCP-7 (Structured Synchronizing Merge)	●	●	○	●	●	○	●	●
WCP-8 (Multi-Merge)	○	◓	◓	○	●	●	○	●
WCP-9 (Structured Discriminator)	○	○	○	○	◓	◓	○	●
WCP-10 (Arbitrary Cycles)	○	○	●	○	●	●	●	●
WCP-11 (Implicit Termination)	●	●	○	●	●	●	●	●
WCP-12 (Multiple Instances without Synchronization)	○	●	●	●	●	●	○	●
WCP-13 (Multiple Instances with a Priori Design-Time Knowledge)	○	●	○	○	●	●	○	●
WCP-14 (Multiple Instances with a Priori Run-Time Knowledge)	○	●	○	○	●	●	○	◓
WCP-15 (Multiple Instances without a Priori Run-Time Knowledge)	○	●	○	○	○	○	○	◓
WCP-16 (Deferred Choice)	○	●	●	●	●	●	○	●
WCP-17 (Interleaved Parallel Routing)	○	◓	●	○	○	○	○	○
WCP-18 (Milestone)	○	◓	●	○	○	○	○	●
WCP-19 (Cancel Task)	○	◓	●	●	●	●	○	●
WCP-20 (Cancel Case)	○	◓	○	●	●	●	○	●
WCP-21 (Structured Loop)	●	●	○	●	●	●	○	●
WCP-22 (Recursion)	●	○	●	○	○	○	○	○

Table 7.3 – Original Workflow Control-flow Patterns WCP-1 to WCP-22

which blocks further input to the activity when an input is received. In order to get further inputs, the contract needs to be reset.

The patterns added later on, left the complete set of control-flow patterns unordered from a conceptual perspective, so we will not be discussing them in numerical sequence. The contracts of CPMF allow the AND-type logical connectors and follow a blocking mechanism, which can be reset (as per the design), so WCP-9 is directly supported. WCP-10, WCP-21 and WCP-22 concern the iteration patterns for the transfer of control. WCP-10 pattern requires the ability of activity to represent cycles of execution. Each CPMF activity specifies a property for defining the number of iterations that it will undergo. This number of iterations can also be changed at runtime. The number of iterations can also be based dynamically on a pre-conditions (that terminate loops), which controls the actual number of iterations at runtime. Each activity can have multiple entry and exit points for the transfer of control. This can also be exploited for ending an execution cycle. Thus a direct support for WCP-10 is present in CPMF. Table 7.4 summarizes the evaluation for patterns after WCP-21. WCP-21 is also supported by CPMF as pre-conditions and post-conditions can determine the continuation of the loop. WCP-22 requires an activity to be able to invoke itself during execution. This recursive behavior of the process model is not yet supported by the CPMF tool implementation.

In order to implement WCP-11, a process model should have a mechanism to terminate an activity that has provided all its artifacts and is not going to produce them anymore. CPMF allows the definition of activity lifecycle, which takes into account all such situations where an activity should change its state to complete, thus providing direct support for this pattern. CPMF also supports WCP-43, where all sub-activities of an activity are terminated, once a parent activity is complete. Multiple instances of an activity are allowed in CPMF, where each instance has its own data elements. These instances can execute concurrently, thus providing support for WCP-12. The exact number of instances can be defined at implementation level, instantiation level and may even be modified during execution. This allows us to rate CPMF with a direct support for WCP-13 and partial support for WCP-14 and WCP15. The two later patterns are evaluated for partial support because CPMF framework does not offer the capability of synchronizing multiple instances of an activity at their completion. However this can be achieved through the pre-conditions of the subsequent activities. All the instances of an activity in CPMF have the same contracts as defined for the activity. These instances trigger different events based on the sources. The subsequent activity after the multiple instance activity can listen to the events based on the event tags or the source activity type. The contract of the subsequent activity can define a complete join for all instances or a partial join for some instances. This allows direct support for WCP-34 and WCP-36. However, WCP-35 is not currently implementable in CPMF because a subsequent activity can not cancel the instances of a previous activity in a sequence of control flow.

The concrete level of CPMF does not define a fixed order of activity execution. It is based on the dependencies of the contracts of activity definitions. So a partial ordering of activities (like in other process models that rely on the ordering of activities) is

Pattern	WebSphere	FlOWer	COSA	BPEL	BPMN	UML	EPCs	CPMF
WCP-23 (Transient Trigger)	○	○	●	○	○	●	-○	●
WCP-24 (Persistent Trigger)	○	●	●	●	●	●	◓	●
WCP-25 (Cancel Region)	○	○	◓	◓	◓	●	○	◓
WCP-26 (Cancel Multiple Instance Activity)	○	○	○	○	●	●	○	◓
WCP-27 (Complete Multiple Instance Activity)	○	◓	○	○	○	○	○	◓
WCP-28 (Blocking Discriminator)	○	○	○	○	◓	◓	○	●
WCP-29 (Cancelling Discriminator)	○	○	○	○	●	●	○	●
WCP-30 (Structured Partial Join)	○	○	○	○	◓	◓	○	◓
WCP-31 (Blocking Partial Join)	○	○	○	○	◓	◓	○	◓
WCP-32 (Cancelling Partial Join)	○	○	○	○	◓	●	○	◓
WCP-33 (Generalised AND-Join)	○	○	○	○	●	○	◓	◓
WCP-34 (Static Partial Join for Multiple Instances)	○	○	○	○	◓	○	○	●
WCP-35 (Cancelling Partial Join for Multiple Instances)	○	○	○	○	◓	○	○	○
WCP-36 (Dynamic Partial Join for Multiple Instances)	○	○	○	○	○	○	○	●
WCP-37 (Local Synchronizing Merge)	●	●	●	●	○	◓	●	●
WCP-38 (General Synchronizing Merge)	○	○	○	○	○	○	○	○
WCP-39 (Critical Section)	○	◓	●	●	○	○	○	○
WCP-40 (Interleaved Routing)	○	◓	●	●	◓	○	○	○
WCP-41 (Thread Merge)	○	○	○	◓	●	●	○	●
WCP-42 (Thread Split)	○	○	○	◓	●	●	○	●
WCP-43 (Explicit Termination)	○	○	●	○	●	●	○	●

Table 7.4 – Extended Workflow Control-flow Patterns WCP-23 to WCP-43

supported by CPMF. However, CPMF can not restrict two activities from executing in parallel, if they have no inter-dependencies. This restricts CPMF to implement WCP-17 and WCP-40. The *life-cycle* contracts of an activity are responsible for notifying/listening to the state of child and parent activities. This allow activities to synchronize their execution accordingly in a hierarchy. This possibility allows us to implement WCP-18. WCP-39 requires an execution branch in the process model to be able to pause another execution branch, where both branches originate from a single source activity. CPMF framework does not restrict such behavior in the process model, but it is not implemented in the CPMF tool, thus we rate this pattern as not supported yet.

A lifecycle description of every activity in CPMF allows the cancellation of primitive and composite activities, thus allowing the implementation of WCP-19 and WCP-20. However for canceling a set of activities in a process, each of them has to be canceled individually. CPMF does not allow selecting a 'non-connected' set of activities based on some criteria for any operation. Cancellation of such a set of activities is possible if they are grouped together in a single process or by defining an automatic primitive activity that sends the cancellation events to all activities in the set. Thus WCP-25 is rated as partial support, as per the evaluation criteria of the pattern. Cancellation of all the activity instances of an activity is also handled individually, thus a partial support for WCP-26 as well. However, due to the lack of direct support for synchronization of multiple instances at their completion, CPMF is not able to provide a direct support. A partial support for WCP-27 can be provided by embedding every activity within an activity sending a terminating event. A subsequent activity can be defined to wait for all these event for synchronization.

The event management system at the concrete level of CPMF process model requires a trigger to activate an activity. This trigger can either be transient or persistent, a choice defined within the contract of the activity. Direct support for WCP-23 is offered due to the support of transient trigger, where the activity starts when it receives the trigger, but can not start later on if it could not start earlier. WCP-24 is supported through the support of persistent trigger, where the triggers are retained until the pre-conditions are met.

Two different types of OR-joins are defined as WCP-28 and WCP-29. WCP-28 implements a blocking connector where the connector is blocked when one of the inputs is received. This connector can be reset by the process. WCP-29 implements this connector in a way that the other inputs are canceled. Both these forms of OR-joins can be implemented in the required contract of an activity in CPMF. Partial AND-joins are not offered through a dedicated construct in CPMF. Though they can also be encoded in the required contracts through the use of conditions. Same is th case for partial AND-joins with blocking behavior and canceling behavior. Thus we rate these patterns (WCP-30 to WCP-33) for partial support, as per the evaluation criteria given with the patterns. The merging connectors are encoded within the contracts of the activity, thus the conditions for merging can be based on some local data or can also be received from some other activity. These conditions of merging may specify how many branches need to be merged. This allows us to implement

WCP-37. WCP-38 can not be implemented by CPMF because an activity (in current implementation) can not guarantee that a non-enabled execution branch will not be enabled in the future and hence some input will not arrive from previous activities in the process. WCP-41 and WCP-42 concern splitting and merging process threads, instead of multiple instances of an activity. All processes in CPMF are contained within a composite activity and a thread of a process corresponds to an instance of a composite activity. Thus both these patterns are supported, the same way multiple instances of primitive activities are supported.

7.4 Workflow Resource Patterns

Workflow Resource Patterns (WRP) define the scenarios to capture the behavior of the process model regarding the representation and utilization of resources. A resource in these patterns is considered as an entity that can perform a work. It can either be human or non-human. Table 7.5 presents a summary of the evaluations for CPMF implementations of first 20 patterns along with the comparisons of other process modeling approaches. These 20 patterns are of two kinds: *creation patterns* that deal with the manner in which the resources are associated to the activities and *push patterns* that capture the situations where newly created activities are proactively associated to resources. The rest of the patterns are summarized in table 7.6. These patterns are *pull patterns* where resources are made aware of associated activities, *detour patterns* dealing with interruptions, *auto-start patterns* for automatic activities, *visibility patterns* that define the scope of the resource and the *multiple resource patterns*.

CPMF framework presents a refinement-based approach for process modeling. The specification phase of the process models specify the associated *responsibilities* and *roles* for each activity. A responsibility assignment matrix for each process model assigns responsibilities to the corresponding roles. The implementation phase process model refines the *roles* as a collection of capabilities. Finally, the instantiation process model associates each role with actors or tools. Actors represent the human performers of the activities, whereas the tools are the hardware/software tools needed to perform an activity. Because the identity of the resource can be specified at the design time, CPMF offers a direct support for WRP-1. WRP-2 is also supported by CPMF because of the role based distribution of the 'work' to the actors. WRP-3 demands the ability to defer the exact specification of actor to runtime. CPMF allows to execute the process models with partial details, where the final details of the process model can either be added/amended to the executing process model. Thus WRP-3 is also supported by our approach.

WRP-4 demands that a range of privileges should be associated to the resources. CPMF offers a direct support for this pattern through the use of responsibilities. Responsibilities associated with each roles for a specific activity, specify the privileges of that role in performing it. For example, for an activity the *technical lead* can have the privilege of authorization. The ability to specify that two primitive activities would be performed by two different actors is demanded by WRP-5. This is supported

Pattern	WebSphere	FlOWer	COSA	BPEL	BPMN	UML	CPMF
WRP-1 (Direct Distribution)	●	●	●	●	●	●	●
WRP-2 (Role-Based Distribution)	●	●	●	●	●	●	●
WRP-3 (Deferred Distribution)	●	○	○	●	○	○	●
WRP-4 (Authorization)	○	●	●	○	○	○	●
WRP-5 (Separation of Duties)	●	●	◓	○	○	○	●
WRP-6 (Case Handling)	○	●	○	○	○	○	●
WRP-7 (Retain Familiar)	●	●	●	●	○	○	●
WRP-8 (Capability-Based Distribution)	○	●	●	●	○	○	●
WRP-9 (History-Based Distribution)	○	○	◓	◓	○	○	◓
WRP-10 (Organisational Distribution)	●	◓	●	◓	○	○	○
WRP-11 (Automatic Execution)	○	●	●	●	●	●	●
WRP-12 (Distribution by Offer - Single Resource)	○	○	◓	●	○	○	○
WRP-13 (Distribution by Offer - Multiple Resources)	●	●	●	●	○	○	○
WRP-14 (Distribution by Allocation - Single Resource)	●	●	●	●	●	●	●
WRP-15 (Random Allocation)	○	○	●	◓	○	○	●
WRP-16 (Round Robin Allocation)	○	○	◓	◓	○	○	●
WRP-17 (Shortest Queue)	○	○	●	◓	○	○	◓
WRP-18 (Early Distribution)	○	●	○	○	○	○	●
WRP-19 (Distribution on Enablement)	●	●	●	●	●	●	●
WRP-20 (Late Distribution)	○	○	○	○	○	○	○

Table 7.5 – Original Workflow Resource Patterns WRP-1 to WRP-20

by CPMF and is carried out once the process model is refined to the appropriate life cycle phase. WRP-6 demands the possibility of assigning a resource to a *case* when it is executed. A *case* in CPMF corresponds to an instance of a composite activity (composed of a process). A composite activity (like primitive activities) can be allocated to the actors at runtime, hence supporting WRP-6. WRP-7 allocates a resource to an activity that performed the last activity. CPMF has the capability to access the last executed activity from it execution log and from that activity, the associated actor of that activity can be accessed. This allows CPMF to implement this pattern. CPMF also offers a mechanism of assigning the resources to an activity based on their capabilities, thus supporting WRP-8. WRP-9 allocates the resources to an activity based on their previous execution history. An execution history of activities is kept by the interpreter. This can be exploited to develop the execution history for a specific actor. Through a little programmatic extension this can be achieved. Thus we rate this pattern for partial support, according to its evaluation criteria. CPMF currently does not support WRP-10, because an actor is associated with an organization, but its position is not specifiable in the organizational hierarchy. Thus actors can not be assigned to activities based on their relationships with other actors. CPMF allows the execution of activities that are not allocated to any actors, thus providing a support for WRP-11.

The current implementation of CPMF prototype does not allow the allocation of resources to the activities on offer basis *i.e.* the assigned actors do not have the choice to approve or reject the allocation. For these reasons, we do not support WRP-12, WRP-13 and WRP-23 where resource allocation is non-binding. All assignments of actors to the activities are considered binding to them. The selection of appropriate actor can be chosen based on the capabilities, randomly or through a cycle between a set of actors, hence supporting WRP-14 to WRP-16. Every actor maintains a list of allocated activities. A little programmatic extension can be used to calculate the working queues of each actor and allocations can be based on it. Thus WRP-17 is partially supported by CPMF. The allocations of actors to the corresponding activities can be carried out at design time or when even when the activity is loaded in the interpreter for execution, thus supporting WRP-18. An actor can be assigned to a task even during the execution of the process (where this activity is not yet executing). This means that the precise primitive activity is assigned with an actor, once its parent activity has started its execution. This primitive activity is enabled for execution but requires the allocation of an actor for execution, thus WRP-19 can be implemented. However, it can not start its execution unless an actor has been allocated to it. Thus a late allocation of resource after execution has started, as required by WRP-20 is not supported in CPMF framework.

WRP-21 requires a resource initiated allocation for the activities. An actor in CPMF can allocate an activity for himself if he has appropriate privileges to do so, thus supporting WRP-21. It is also possible for an actor to provide an internal trigger for the execution of the activity. This gives an actor the authority to trigger an activity without any external event. Thus pattern WRP-22 is also supported by CPMF implementation. The work queue of an actor is initially ordered by the reception of events. In case multiple activities are allocated to an actor that have

Pattern	WebSphere	FlOWer	COSA	BPEL	BPMN	UML	CPMF
WRP-21 (Resource-Initiated Allocation)	○	●	◓	○	○	○	●
WRP-22 (Resource-Initiated Execution - Allocated Work Item)	●	●	●	●	○	○	●
WRP-23 (Resource-Initiated Execution - Offered Work Item)	●	○	●	●	○	○	●
WRP-24 (System-Determined Work Queue Content)	○	●	○	○	○	○	●
WRP-25 (Resource-Determined Work Queue Content)	●	●	●	●	○	○	●
WRP-26 (Selection Autonomy)	●	●	●	●	○	○	●
WRP-27 (Delegation)	●	○	●	●	○	○	●
WRP-28 (Escalation)	●	○	●	●	○	○	●
WRP-29 (Deallocation)	○	○	●	●	○	○	●
WRP-30 (Stateful Reallocation)	●	○	●	●	○	○	●
WRP-31 (Stateless Reallocation)	○	○	○	○	○	○	●
WRP-32 (Suspension-Resumption)	◓	○	●	●	○	○	●
WRP-33 (Skip)	●	●	●	●	○	○	●
WRP-34 (Redo)	○	●	○	○	○	○	◓
WRP-35 (Pre-Do)	○	●	○	○	○	○	●
WRP-36 (Commencement on Creation)	○	○	●	●	●	○	○
WRP-37 (Commencement on Allocation)	●	○	○	○	○	○	●
WRP-38 (Piled Execution)	○	○	○	○	○	○	◓
WRP-39 (Chained Execution)	○	●	○	●	●	○	◓
WRP-40 (Configurable Unallocated Work Item Visibility)	○	○	○	○	○	○	○
WRP-41 (Configurable Allocated Work Item Visibility)	○	●	○	○	○	○	●
WRP-42 (Simultaneous Execution)	●	◓	●	●	●	●	●
WRP-43 (Additional Resources)	○	○	◓	●	○	○	●

Table 7.6 – Workflow Resource Patterns WRP-21 to WRP-43

no dependencies between them and have flexible deadlines, then the actor has the possibility to re-arranging his working queue and selecting the activity that he wants to perform first, thus supporting WRP-23 to WRP-26.

CPMF framework allows to manipulate the activities during the execution of a process model. Thus an activity allocated to one actor can be re-allocated to another actor in an executing process model. WRP-27 requires an actor to be able to re-allocate his activity to another actor. CPMF offers the possibility to do so, only if the actor has enough privileges to do so. Allocation/reallocation of activities in a process is normally carried out by process owner. But rights to do such actions can be given to the actors of the activities. The same criteria goes for the de-allocation of an activity from an actor. CPMF process implementation tool provide support to implement WRP-27 to WRP-29. The re-allocation of activities to other users during executing is carried out as activity adaptation. The state of the activity can be transferred to the new activity in such situations. This allows us to implement WRP-30. WRP-31 that requires a stateless transfer but the activity has to use the interactive method of state transfer between the activities. In this mechanism, a process owner can choose to link no properties between the new activity and the replaced activity. Hence no state is transfered in this case. WRP-32 and WRP-33 are also supported by CPMF framework, as the actor has the possibility to suspend and resume the execution of an activity and to mark it as complete (even if the activity is skipped). An actor can also redo an activity, if the activity lifecycle allows so. However the subsequent activities after this activity will not be repeated unless their life-cycles also permits to start over an activity after completion. The default lifecycle provided by the tool implementation does not permit that. Thus we rate WRP-34 for a partial support.

Once an activity has been allocated to an actor, it is presented to him on the project dashboard. If this activity has no dependencies over other artifacts, an actor can perform this activity before its scheduled plan and even upload the artifacts in the repository, thus supporting WRP-35 and WRP-37. However CPMF does not allow the execution of an activity by an actor before it is allocated to him/her, thus providing no support for WRP-36. Once an actor has completed one task, next activity in his queue can be triggered automatically or manually, if all the dependencies of those activities are met. When an activity requires an external trigger, then it will not start automatically. If the external trigger is to be initiated by an actor, then the activity should be started manually. If it does not require a trigger, it starts as soon as the previous activity is complete (and all other pre-conditions are met). Multiple instances of an activity are also executed the same way. Thus CPMF allows partial support to implement the patterns WRP-38 and WRP-39, where little programmatic extensions are required.

An actor in a CPMF process model is able to access the activities allocated to him through the project management dashboard. No actor (other than process administrator) can access the activities that are not allocated to him/her. Thus CPMF does not support WRP-40. However WRP-41 is supported, which permits to configure the scope of allocated activities to the actors. CPMF allows the execution of multiple

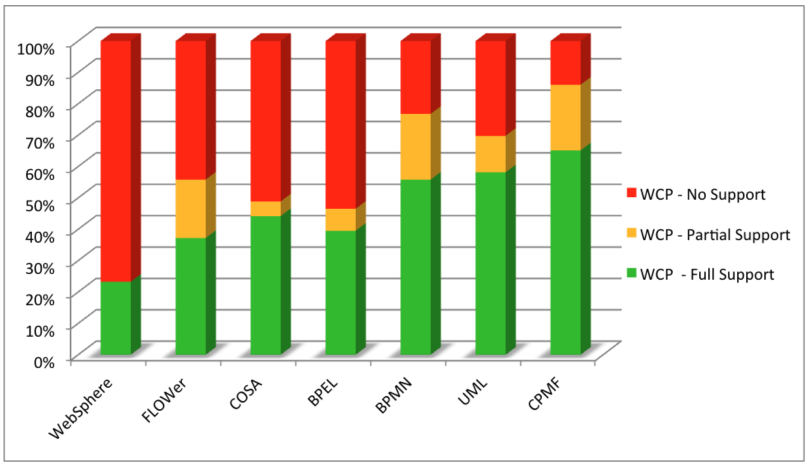


Figure 7.2 – Support for Workflow Control-flow Patterns

activities by the same actor simultaneously, thus supporting WRP-42. As explained earlier, an activity can be adapted during it execution. Thus, if additional roles need to be attached to an activity or a role needs to be played by more actors than actually assigned, this can be handled through activity adaptations. This activity is adapted to a new activity with the required properties. The state is transferred between the old activity and the activity replacing it. This allows use to implement WRP-43.

7.5 Discussion

Workflow Patterns capture different aspects of process modeling languages. The workflow pattern initiative by Eindhoven University of Technology and Queensland University of Technology offers patterns for control-flow, resources, data and exception handling. Even though they primarily focus on the evaluations of different workflow languages, an informal equivalence of workflows is possible in other process modeling languages. Different process modeling approaches (including other than workflows) are evaluated and their rating are published on the workflow initiative website[3]. Not all of the approaches presented in the state of the art for this thesis are evaluated. We have provided the evaluations of CPMF framework and compared it with other approaches (where the data is taken from the workflow initiative website). These evaluations concern the data, resource and control-flow patterns only. CPMF is not currently providing support for exception handling, so exception handling patterns are not implemented and are not included in this chapter.

CPMF framework supports most of the Workflow control-flow patterns either completely or partially, as shown in the figure 7.2. Workflow control-flow patterns focus on capturing the control-flow mechanisms provided by a software process modeling approach. The basic control flow patterns concern the availability of logical connectors to route the control flow among activities. CPMF framework allows the

3. http://www.workflowpatterns.com/

encoding of this control flow logic inside activity contracts. The split connectors are embedded inside the provided contracts, where as the merge connectors are encoded inside the required connectors. Other process modeling approaches (that focus on the flow of data/control) explicitly represent these logical connectors outside the activities, within the process. CPMF on the other hand focuses on other issues related to process modeling and does not want to put unnecessary focus on the routing information. The main focus of CPMF is to target completeness of individual activities where a refinement based approach is presented. For the interactions of activities, CPMF focuses on defining it in a bi-layered approach and through the use of contractual paradigm. The motivation of encoding these logical connectors within the contracts was to put more attention to the core issues discussed by CPMF. The process modeling language for CPMF can always be extended to explicitly represent these logical connectors in a process, outside the activities.

Out of the workflow control patterns that are not supported by CPMF the main reasons stem back mostly to the support of recursion and default activity life cycles. Recursion of an activity is not implemented in the tool support. Though, an activity can invoke its own instance at the runtime, but managing the lifecycle of the two activities to get the recursive behavior would need addition support for such linking. Multiple instances of an activity can execute in an executing process, but they are considered as parallel or sequential executions in the same context. This does not mean that it is impossible to achieve it. The amount of effort required to implement this pattern rates it as having no support. A default activity life cycle is provided by CPMF for the instant execution of activities. The framework allows its users to define different lifecycles for different activities. But for this evaluation, we have considered the default lifecycle only. Many of the control-flow patterns which are not supported currently, can be supported by defining appropriate life-cycles for the concerned activities.

Workflow data patterns focus on the definition and management of artifacts in the process model. CPMF framework defines artifact contracts for each activity at the abstract level and artifact specification, artifact metamodel and artifact lifecycle state-machine are used to describe the structure and behavior of each artifact in detail. Moreover, an artifact repository at the concrete level manages the transfer of artifacts between multiple activities. This artifact repository manages the concurrent access of different activities and multiple versions of the artifacts. Most of the patterns in this category are implementable by CPMF framework either completely or partially, as shown in figure 7.3. Two patterns that are not supported by CPMF concern the transfer of data elements between the activities based on their values instead of references. Some of the data elements can be transferred between the activities based on their value (through messages). However, for the transfer of artifacts between the activities, their repository url is used.

Workflow resource patterns define the manner in which resources (human and non-human) are represented and utilized by the process modeling approach. Most of the necessary constructs used by these patterns are defined by the process modeling approach. However, some patterns are not implementable by the tool support pro-

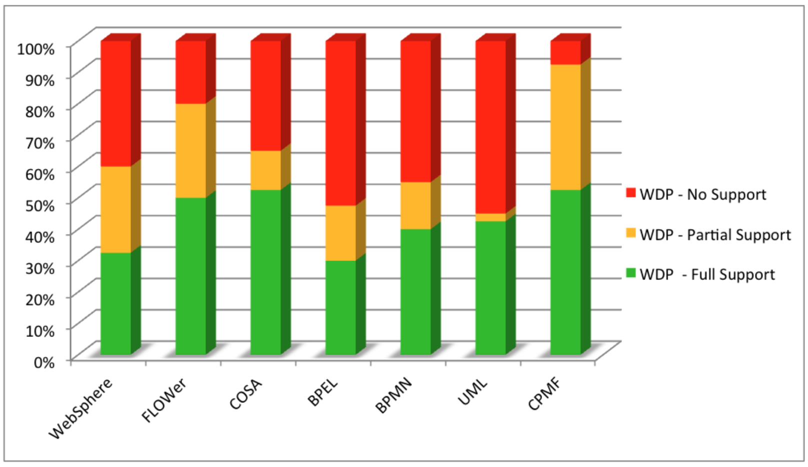


Figure 7.3 – Support for Workflow Data Patterns

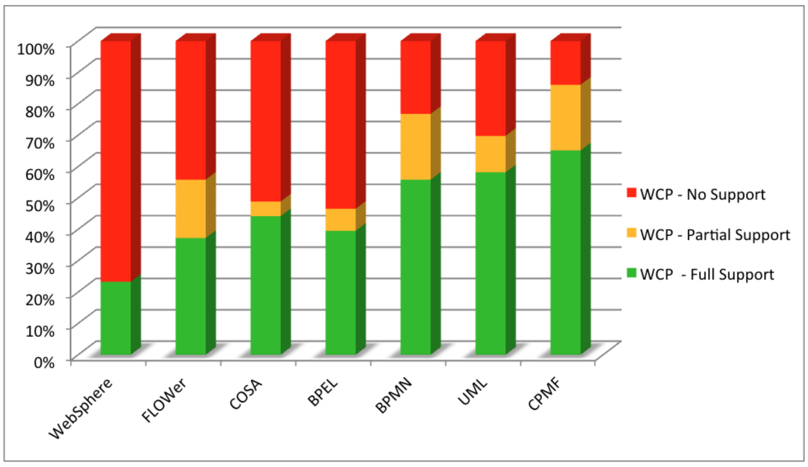


Figure 7.4 – Support for Workflow Resource Patterns

vided along with CPMF framework. Constructs that are missing in CPMF process model are the properties related to the organizational hierarchy of the actors that can define their inter-relationships. Patterns that are not implemented due to the tool implementations concern the support for offer based allocation and the execution of unallocated activities. CPMF tool support does not provide support for offer based resource allocation such that the resource has the privilege to accept or reject the duties assigned to him/her. The tool also does not support the execution of activities by an actor that is not associated to the activities (process administrator being an exception). CPMF support for workflow resource patterns is better from the rest of the approaches in state of the art, as shown in figure 7.4.

Contrary to other process modeling products/languages that are mostly commercial, CPMF tool support is a prototype to evaluate the implementation feasibility of the language. Thus it does not provide a complete implementation of the language yet. This also resulted in a partial support or unavailability of support for different workflow patterns. With this argument, we want to put forward the fact that the process modeling language proposed in this framework is not hindering the implementation of any of these patterns. If some workflow patterns are not supported by the CPMF framework, it is the shortcoming of the tool implemented for CPMF, not the process modeling language itself. Despite these shortcomings of the prototype tool, the results of implementing the workflow patterns are encouraging. Table 7.7 summarizes the number of patterns supported by CPMF in comparison with other approaches in state of the art.

Pattern	WebSphere	FlOWer	COSA	BPEL	BPMN	UML	**CPMF**
WDP Direct Support	13	20	21	12	16	17	**21**
WDP Partial Support	11	12	5	7	6	1	**16**
WDP Support Missing	16	8	14	21	18	22	**3**
WCP Direct Support	10	16	19	17	24	25	**28**
WCP Partial Support	0	8	2	3	9	5	**9**
WCP Support Missing	33	19	22	23	10	13	**6**
WRP Direct Support	19	22	24	24	8	6	**30**
WRP Partial Support	1	2	6	5	0	0	**6**
WRP Support Missing	23	19	13	14	35	37	**7**

Table 7.7 – Workflow Patterns support summary

Part IV

Epilogue

Chapter 8

Conclusion and Perspectives

Contents

Abstract **-** *We conclude our work in this chapter. The contributions of this research work are presented using the solution criteria defined in the first chapter. Finally, limitations of the current work and the possible future prospects are outlined.*

8.1 Contributions and Achievements

The core objective of this thesis was to develop a comprehensive and consistent approach for process modeling that is capable of handling these processes in various stages of their development life cycle. This objective was to be achieved in a way that instead of using "one model fits all phases" approach, multiple models should be developed according to the precise nature of each phase of process development. It was also required to separate the concerns related to data-flow from the control flow such that each concern can be analyzed, developed, maintained and updated individually.

This goal has been satisfied at three levels. First, the fundamental components of business processes have been categorized according to their relevance to the specific phase of process development. These concepts are then used to develop multiple metamodels, each pertaining to a specific process development lifecycle phase. Second, a bi-layered approach has been chosen to separate the data-flow of the process models from their control-flow. A mapping between both these layers guarantees the conformance of control-flow to the specified data-flow. Finally, the development of a prototype implementation equipped with a process interpreter that is responsible for executing the processes and allowing the capability to monitor them. This demonstrates the support for refinement of the processes from specification to their execution.

Seven solution criteria were defined at the beginning of this research, as a means of evaluating the effectiveness of the proposed solution. We revisit each of these criteria to explain how effectively the proposed solution has managed to solve the identified problems in the existing approaches. We update table 3.1 with the evaluations of CPMF regarding each of the solution criteria and present it in table 8.1.

Completeness: One of the main goals of CPMF approach was to develop an approach that caters for all the different phases of process development lifecycle. This means that it had to take into account all the relevant components of a process model pertaining to each phase of process development lifecycle. Contrary to other approaches that focus on a single or some of the phase of process development, CPMF presents the concepts that encompass the complete lifecycle of a process. These concepts are placed into relevant models of the specific phase of process development. For example, the concept of state is important for the activity, only after the implementation phase. Thus the specification level process model is not polluted with this concept. However, when the process model refines to the appropriate phase, it has access to all the relevant concepts.

The notions related to the intent of each activity like goals, objectives and intentions are added to each activity. On one hand they help in providing a comprehensive description of an activity and on the other hand they serve as a guidance for the implementation of the activities. The process developer of a specification level activity might be different from the developer that develops activity implementations. So, they help in guiding the implementation of each activity also. They are also useful when choosing an activity for reuse from the process repository. The use of tags and contracts helps in searching existing process components and these intent related notions help in refining that search.

Because most of the process modeling approaches target one or some of the phases of process development, they either miss out on providing the support for execution or the support for abstract level process specifications. For example, SPEM and EPCs do not provide a direct execution support. Most of the other approaches that tend to enrich them with execution semantics, also end up with providing a transformation to other languages that provide execution semantics like BPEL or XPDL. CPMF framework provides the direct execution support for its process models. The behavior of a process model is defined through the activity and artifact state-machines. Together they allow to develop an executable process modeling whose execution behavior is customizable. The formal semantics of CPMF itself are not defined as yet. A mapping to Hierarchical Petri Nets is developed to specify the interaction behavior of the processes [Golra 12b]. But a formal validation of this semantics is missing.

Due to the completeness of constructs provided by CPMF, it is possible to transform a process model developed with BPMN, BPEL or SPEM *etc.* to a specific phase model in CPMF framework. However, we are afraid that a transformation in opposite direction might result in a conceptual and structural loss. These transformations are not developed as yet, but this argument is based on the richness of concepts offered by CPMF.

Criteria	SPEM	xSPEM	MODAL	BPMN	BPEL	EPCs	YAWL	Little-JIL	CPMF
Completeness									
Architectural constructs	+/-	+	+	+/-	+/-	+/-	+	+	+
Process intents	+/-	+/-	+	+	+/-	-	+/-	+/-	+
Process behavior	-	+	+	+/-	+	+/-	+	+	+/-
Team Development									
Choreography	-	-	-	+	+	-	-	-	+
Task allocation	+	+	+	+	+/-	+	+	+	+
Responsibility assignment	-	-	-	-	-	-	-	-	+
Distributed process development	-	-	-	+/-	+	+	+	+	+
Reusability									
Approach-based systematic	+/-	+/-	+	-	+/-	-	-	+/-	+
Implementation-based systematic	+/-	+/-	+/-	+/-	+	+/-	+/-	+/-	+
Opportunistic	+	+	+	+	+	+	+	+	+
Abstraction									
Phase-wise refinement	-	-	+/-	-	-	+/-	-	-	+
Internal conformance	-	-	-	-	+/-	-	-	-	+
Modularity									
Hierarchical modularity	+	+	+	+	+/-	+	+	+/-	+
Contextual modularity	+/-	+/-	+	+/-	+	-	-	+/-	+
Tailorability									
Static process tailoring	+	+	+	+	+/-	+/-	+/-	+/-	+
Dynamic adaptations	-	-	-	-	+	+/-	+/-	-	+
Enactability									
Direct execution support	-	+	+	-	+	-	+	+	+
Activity lifecycle	-	+/-	+/-	+/-	+/-	-	+/-	+/-	+
Artifact lifecycle	-	-	+/-	-	-	-	-	-	+

Table 8.1 – Evaluation of existing approaches based on the solution criteria

Team Development: With increasing trends of outsourcing and sub-contracting, software development processes are becoming distributed in nature. Moreover, the geographical separation of different actors, teams and departments require an effective support for process development methodologies that allow distributed development of process models. This is supported by the implementation architecture of CPMF, through the use of process repositories that allow concurrent version management. This helps in developing the process models by different teams that are geographically separated. Apart from the distributed development of process models, a support for managing distributed software development processes is also important for a process modeling approach. This requires an effective mechanism to associate work items with resources that would be need to perform it. These resources can either be human or non-human. A stress on interactions between these resources is required to support distributed software development projects.

CPMF framework provides the concepts of responsibility and roles at the specification level. A role is a collection of capabilities to perform certain task and it is assigned with some responsibility regarding the concerned activity. This responsibility defines the privileges of a role related to some activity. These roles are played by actor (humans) and tools (non-humans). A mechanism of interactions among the roles of different activities is presented in terms of message contracts. These message contracts ensure the responsibility of a role to carry out an interaction. Effective means of binding these resources for interactions and modeling them is not common in many other approaches like SPEM, EPCs & YAWL. BPMN models when executed using BPEL also lack the capability of guaranteeing such interactions because of the inability of BPEL to model human processes effectively. These approaches do not provide the mechanisms of integrating responsibility assignment matrices with associated roles of activities. CPMF framework provides this capability for an effective management of resources for a software development project.

Abstraction: Software process models can benefit from the use of abstractions in two ways: *horizontal*, in terms of the advancing phase of process development lifecycle and *vertical* within a single process model to structure it in multiple abstraction layers. Abstraction across multiple phases of process development are exploited by CPMF through the use of multiple metamodels, each pertaining to a specific phase of process development lifecycle. These metamodels have a refinement relationship between them, which serves to refine a process model from the earlier phases of process development till its execution, possibly passing through different phases. The current implementation of CPMF demonstrates this philosophy through the use of three metamodels for specification, implementation and instantiation phases of process development respectively. A process model in instantiation phase is ready for execution by the process interpreter.

The use of abstraction within a process model (of a specific phase in CPMF) can be exploited to develop it in multiple layers. CPMF uses a bi-layered approach to model its processes, where the abstract level is used to specify the data-flow of the process model and the concrete level (conforming to abstract level) defines its control-flow. This approach allows to analyze, develop, maintain and update both these levels separately. Apart from the structural benefits of this separation, it allows to introduce variability in the process model. It also allows to develop the process models in a manner which promotes standardization of the processes. This support of standardization stems from the capability of CPMF to keep the process standard at the abstract level of the process model and develop multiple implementations at the concrete level. The conformance relationship between the two levels guarantees the compliance to adopted standards.

Modularity: Modularity of software process models has been targeted by the process modeling approaches in only one direction *i.e.* hierarchical modularity. This comes from the fact that process models are hierarchical in nature, where all processes are made up of smaller processes that are in turn made up of activities containing multiple tasks. But there is a second dimension to modularity, which accounts for effective partitioning of the process within the same context *i.e.* contextual modularity. This is the generally known type of modularity in software systems that has been a motivation for component-based paradigm. Inspired from the same paradigm, CPMF takes each activity as an equivalent to a process component. Each activity is properly encapsulated, where all interactions (to and from it) are restricted through the defined interfaces. Interfaces of an activity are defined through abstract contracts and concrete contracts depending upon the abstract/concrete level of the process model. Inspirations from Design by Contract (DbC) allowed to integrate pre-conditions and post-conditions to the contracts of the activities. This guarantees the correctness of interaction between two activities. It also allows to decouple the activities by focusing on complete definitions for each activity.

Reusability: One of the benefits of choosing a modular approach for process modeling is that it favors reusability. Other software process approaches in state of the art do not follow the "design to reuse" philosophy and end up with offering only the opportunistic reuse of process fragments. Contrary to them, CPMF follows the design to reuse approach to foster process models that can benefit from the systematic reuse of process fragments. From a process fragment, we mean to say, a part of process model that can be as little as a primitive activity to as big as the complete process model. A part of support for the process model approach is dependent upon the implementation of that methodology. For example, one implementation tool for BPMN can offer the facility to store and retrieve process components from a repository and the other may not. This has got nothing to do with the design choice of the modeling methodology. Along with the choice to follow design to reuse, CPMF implementation tool also supports reusability by offering a process repository. Activities (both abstract and concrete) are stored in the repository for their potential reuse.

Tailorability: Another benefit related to the adoption of modular approach is the support for tailorability. The motivation behind tailoring a software process is to adapt it to the current requirements or to support process improvement. When a support for process improvement is the target, one normally tailors the process model statically. By static tailoring, we mean to say that concerned processes that need to be updated are not executing. This is the most common form of process tailorability. It is supported by CPMF framework in a more effective way in comparison to other approaches. A modular approach with defined interfaces and the use of a bi-layered architecture makes such tailoring effective. The contracts for any new activity (or the update to an existing activity) need to conform to the defined interfaces of corresponding activity definition at the abstract level. This ensures the correctness of the process when tailoring it.

When the goal of process tailoring is to update a process in execution such that it can meet the current requirements, dynamic updates are applied. This dynamic updates replace the process (or a part of it) with the updated process fragments. So, if an activity needs to be replaced by another activity during runtime, this would be handled through dynamic update. CPMF framework allows to dynamically update the executing process. This update may concern adding/removing news instances of an activity that supports multiple instances. It can also replace one activity with another, such that the state of the activity being replaced is transferred to the new activity.

Enactability: For a process modeling approach that can handle the processes in different phases of their development lifecycle, it is important to end up with an executable model. A CPMF process model at the instantiation phase is capable of being enacted directly. The tool support provided with the framework is equipped with a process interpreter. It can bootstrap the process model and execute the activities contained within the process. A project management dashboard is a web interface used to interact with executing processes. Actors that are responsible to carry out the activities log in to this dashboard and can update the state of the executing processes. This interpreter also links the activities to the software development environment. The roles associated with an activity may be played by tools. Thus tool invocations is also handled through this interpreter.

Different process modeling approaches allow to enact the process models. One of the goals targeted by CPMF is also to make the approach flexible enough to handle different complex scenarios. Contrary to other approaches, CPMF framework allows the support to define state-machines for concerned activities and artifacts in a process model. This gives flexibility to the the process developer to define various lifecycles depending upon the precise nature of the process being modeled.

8.2 Limitations and Prospects

The research work carried out for this thesis proposed a new approach for modeling software development processes by giving pivotal focus to the notion of phase-wise process refinement. For process models in a specific phase, inspirations were taken from the concepts of "Design by Contract" and "Design for reuse". We have tried to achieve the goals setup in the beginning of this thesis and have proposed our methodology, Component-oriented Process Modeling Framework (CPMF) along with a prototype implementation of this approach. Just like the concept of "continuous process improvement" in process development, all approaches need to be improved continuously to achieve excellence. This research work has some limitation, which open up room for further improvements. Besides this, it opens up some new dimensions that can be explored in this domain.

— **Formal semantics:** The definition of the proposed process modeling framework is carried out using multiple metamodels. These metamodels define the structure of the process models that are developed using CPMF. Its behavior is explained in the thesis using informal natural language. Implementation of the approach in a prototype evaluates the validity of the approach to a certain level. Formalization of the runtime behavior of the processes can concretize the approach theoretically. A translation of interaction control behavior of CPMF process models to Hierarchical Petri Nets (HPN) has already been done [Golra 12b]. However, this needs a formal validation of the semantics.

— **Prototype limitations:** The prototype implemented alongside this thesis was developed with the intention to demonstrate the feasibility of the approach. This prototype is made up basic components like process editor, process interpreter, artifact & process repositories and a web interface to interact with the executing processes. Many of the process patterns (discussed in chapter 7) that are not currently supported by CPMF are due to the minimal implementation of the prototype. For example, the workflow resource pattern WRP-12 & 13 that concern the allocation of actor to the activities based on offers that they can accept or reject is not implemented. This is not a shortcoming of the process modeling approach. We have focused on demonstrating only the core aspects of the methodology in the prototype. Further extensions to this prototype will result in implementing different scenarios which are inherently supported by the process modeling approach. Similarly, extraction of different types of view from process models can also be implemented in the prototype to extend its functionality.

— **Dynamic updates at abstract level:** CPMF framework allows to dynamically update an executing process. These updates concern the activity implementations at the concrete level of the process model. Activity definitions at the abstract level of the process model are not subject to dynamic updates as yet. Further extensions to the framework can look into the prospects of dynamic updates of the activity definitions. Currently the CPMF only allows a controlled adaptation between already developed activity implementations. Dynamic updates to the abstract level activity definitions will open up new horizons, where dynamic updates to the process will not more be restricted by a controlled set of activity implementations.

— **Multi-layered process modeling** CPMF framework uses a bi-layered architecture for implementation and instantiation phase process models. A multi-layered modeling approach like Lazy Initialization Multilayered Modeling (LIMM) [Golra 11] that can benefit from unlimited number of abstraction layers within a model can be exploited for process modeling. A recent standard has used another multi-level modeling approach (Powertype based metamodeling) for situational method engineering [ISO/IEC 08c]. Implementing multi-layered metamodeling approaches can result in more flexible process modeling approaches that benefit from partial instantiations of the process models at specific level of metamodeling.

— **Support for Business Activity Monitoring (BAM)** The current focus of this thesis remained on the development aspects of the process lifecycle. The possibility to execute the processes using process interpreter is provided by the tool implementations. However, further extensions of this approach in BAM can result in an effective iterative process lifecycle that can allow process improvement more effectively. Every instantiation activity in the process instance model keeps a trace to the previous models till the process specifications. Thus, monitoring an executing process can result in precise recommendations for process improvement. For example, a project manager notices that a lot of multiple instances of a particular instantiation activity are used in the project. This activity allows to trace back to its process specifications. This information can result in improving this activity from specification to instantiation as another development cycle of the process. This will also support continuous process improvement, as the instantiation activities can be replaced at runtime.

— **Comprehensive software development Support** The graphical process editor for CPMF framework is based on the viewpoint concept from Openflexo. To demonstrate the support for automatic activity execution, we have used some unit test examples. They serve fine to demonstrate the linkage of process modeling framework to the associated software development environment. However, a serious integration of the proposed process modeling framework with Openflexo can result in a technology that can be used for real life software process development. This can be achieved through the integration of the CPMF process interpreter with Openflexo software development framework in a way that automatic processes can invoke and interact with a set of tools handled by Openflexo. Openflexo already provides this support, built around BPMN. Adding CPMF interpreter to this open source framework can make this research project usable by the industry.

— **Real life process modeling** We have implemented ISPW benchmark for process modeling in this thesis to demonstrate the key features of the approach. A case was also carried out for modeling some pseudo-real test processes. But both these processes are modeled so as to show the strengths and weaknesses of the proposed framework. Implementation of real life processes in an industry or for some student projects can be carried out. Surveys based on these real life process implementations can be fruitful to guide future perspectives of this research project.

Part V

Bibliography and appendices

Bibliography

[Adams 05] Michael J. Adams, Arthur H.M. ter Hofstede, David Edmond & Wil M.P. van der Aalst. *Facilitating Flexibility and Dynamic Exception Handling in Workflows through Worklets.* In Orlando Bello, Johann Eder, Oscar Pastor & Joao Falcao e Cunha, editors, Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05) Forum, pages 45–50, Porto, Portugal, 2005. FEUP Edicoes. 2, 3

[Adams 06] Michael J. Adams, Arthur H.M. ter Hofstede, David Edmond & Wil M.P. van der Aalst. *Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows.* In Robert Meersman & Zahir Tari, editors, On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. Springer Berlin Heidelberg, 2006. 5

[Almeida da Silva 11] M.A. Almeida da Silva, R. Bendraou, J. Robin & X. Blanc. *Flexible Deviation Handling during Software Process Enactment.* In Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011, pages 34–41, 2011. 32, 34

[Ambriola 97] Vincenzo Ambriola, Reidar Conradi & Alfonso Fuggetta. *Assessing process-centered software engineering environments.* ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 6, no. 3, pages 283–328, jul 1997. 20

[Andersen 01] Bjørn Andersen & Tom Fagerhaug. *Advantages and disadvantages of using predefined process models.* Strategic Manufacturing: IFIP WG5, 2001. 34

[Armbrust 09] Ove Armbrust, Masafumi Katahira, Yuko Miyamoto, Jürgen Münch, Haruka Nakao & Alexis Ocampo. *Scoping software process lines.* Software Process: Improvement and Practice, vol. 14, no. 3, pages 181–197, 2009. 22

[Atkinson 02] Colin Atkinson & Thomas Kühne. *Rearchitecting the UML infrastructure.* ACM Transactions Modeling and Computer Simulations, vol. 12, no. 4, pages 290–321, October 2002. 73

[Bandinelli 94] S. Bandinelli, M. Braga, A. Fuggetta & L. Lavazza. *The architecture of the SPADE-1 Process-Centered SEE.* In Brian C. Warboys, editor, Software Process Technology, volume 772 of *Lecture Notes in Computer Science*, pages 15–30. Springer Berlin Heidelberg, 1994. 20

[Beck 99] Kent Beck. *Embracing change with extreme programming.* Computer, vol. 32, no. 10, pages 70–77, 1999. 36

[Belkhatir 91] N. Belkhatir, J. Estublier & W.L. Melo. *A Support to Large Software Development Process.* In Proceedings of the First International Conference on the Software Process, 1991, pages 159–170, 1991. 20

[Bendraou 07] Reda Bendraou, Benoit Combemale, X. Cregut & M.-P. Gervais. *Definition of an Executable SPEM 2.0.* In Proceedings of the 14th Asia-Pacific Software Engineering Conference, 2007. APSEC 2007, pages 390–397, 2007. 32, 41, 42, 43, 46, 76, 202

[Beugnard 99] Antoine Beugnard, Jean-Marc Jezequel, Noël Plouzeau & Damien Watkins. *Making components contract aware.* Computer, vol. 32, no. 7, pages 38–45, 1999. 23, 24

[Bocchi 10] Laura Bocchi, Kohei Honda, Emilio Tuosto & Nobuko Yoshida. *A Theory of Design-by-Contract for Distributed Multiparty Interactions.* In Paul Gastin & François Laroussinie, editors, CONCUR 2010 - Concurrency Theory, volume 6269 of *Lecture Notes in Computer Science*, pages 162–176. Springer Berlin Heidelberg, 2010. 23, 24

[Boehm 86] B Boehm. *A spiral model of software development and enhancement.* SIGSOFT Software Engineering Notes, vol. 11, no. 4, pages 14–24, August 1986. 36

[Boehm 96] Barry Boehm. *Anchoring the software process.* Software, IEEE, vol. 13, no. 4, pages 73–82, 1996. 7, 59

[Booch 97] Grady Booch, James Rumbaugh & Ivar Jacobson. *UML notation guide, version 1.1.* Rational Software Corporation, Santa Clara, CA, 1997. 2

[Börger 12] Egon Börger. *Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL.* Software & Systems Modeling, vol. 11, no. 3, pages 305–318, 2012. 59, 60, 160

[Bruynooghe 91] R. F. Bruynooghe, J. M. Parker & J. S. Rowles. *PSS: A System for Process Enactment.* In Proceedings of the First International Conference on the Software Process, pages 128–141, 1991. 2

[Cass 99] Aaron G Cass, Barbara Staudt Lerner, Eric K McCall, Leon J Osterweil & Alexander Wise. *Logically central, physically distributed control in a process runtime environment.* Technical Report UM-CS-1999-065, University of Massachusetts, Computer Science Department, Amherst, 1999. 63

[Cass 00] A.G. Cass, A.S. Lerner, E.K. McCall, Leon J. Osterweil, Stanley M. Sutton Jr. & Alexander Wise. *Little-JIL/Juliette: a process definition language and interpreter.* In Proceedings of the 2000 International Conference on Software Engineering, 2000., pages 754–757, 2000. 61, 202

[Céret 13] Eric Céret, Sophie Dupuy-Chessa, Gaëlle Calvary, Agnès Front & Dominique Rieu. *A taxonomy of design methods process models.* Information and Software Technology, vol. 55, no. 5, pages 795 – 821, 2013. 36

[Chang 01] E. Chang, E. Gautama & T.S. Dillon. *Extended activity diagrams for adaptive workflow modelling.* In Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001. ISORC - 2001, pages 413–419, 2001. 5

[Chung 89] Eun K. Chung. *A survey of Process Modeling Tools.* Technical Report No. 7, Computer Integrated Construction Research Program, The Pennsylvania State University, 1989. 2

[Coalition 99] Workflow Management Coalition. *The Workflow Management Coalition Specification: Terminology & Glossary.* Technical Report No. WFMC-TC-1011, Issue 3.0, feb 1999. 29, 32

[Committee 02] ESD Symposium Committee. *ESD Symposium Committee Overview: Engineering Systems Research and Practice.* Technical Report No. ESD-WP-2003-01.20, Massachusetts Institute of Technology Engineering Systems Division, may 2002. 21

[Cortes-Cornax 12] Mario Cortes-Cornax, Alexandru Matei, Emmanuel Letier, Sophie Dupuy-Chessa & Dominique Rieu. *Intentional Fragments: Bridging the Gap between Organizational and Intentional Levels in Business Processes.* In Robert Meersman, Hervé Panetto, Tharam Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi & Isabel F. Cruz, editors, On the Move to Meaningful Internet Systems: OTM 2012, volume 7565 of *Lecture Notes in Computer Science*, pages 110–127. Springer Berlin Heidelberg, 2012. 8, 36

[Crnkovic 06] Ivika Crnkovic, Michel R. V. Chaudron & Stig Larsson. *Component-Based Development Process and Component Lifecycle.* In International Conference on Software Engineering Advances, page 44, 2006. 9

[Curtis 92] Bill Curtis, Marc I. Kellner & Jim Over. *Process modeling.* Communications of the ACM, vol. 35, no. 9, pages 75–90, sep 1992. 2, 18, 19

[Davenport 93] Thomas H. Davenport. Process innovation: reengineering work through information technology. Harvard Business School Press, Boston, MA, USA, 1993. 18

[Diaw 11] Samba Diaw, Rédouane Lbath & Bernard Coulette. *Specification and Implementation of SPEM4MDE, a metamodel for MDE software processes.* In International Conference on Software Engineering and Knowledge Engineering, pages 646–653, 2011. 44, 202

[Dowson 94] Mark Dowson & Christer Fernström. *Towards requirements for enactment mechanisms.* In Brian C. Warboys, editor, Software Process Technology, volume 772 of *Lecture Notes in Computer Science*, pages 90–106. Springer Berlin Heidelberg, 1994. 20

[Eclipse 13] Project Eclipse. *Stardust - Comprehensive Business Process Management for Eclipse*, june 2013. 6, 30

[Efftinge 06] Sven Efftinge & Markus Völter. *oAW xText: A framework for textual DSLs.* In Workshop on Modeling Symposium at Eclipse Summit, volume 32, 2006. 118

[Elias 12] Mturi Elias & Paul Johannesson. *A Survey of Process Model Reuse Repositories.* In Sumeet Dua, Aryya Gangopadhyay, Parimala Thulasiraman, Umberto Straccia, Michael Shepherd & Benno Stein, editors, Information Systems, Technology and Management, volume 285 of *Communications in Computer and Information Science*, pages 64–76. Springer Berlin Heidelberg, 2012. 3, 71

[ESA-ESTEC 09] ESA-ESTEC. *ECSS-E-ST-40C, Space engineering - Software.* Requirements & Standards Division, ESA-ESTEC, mar 2009. 66, 67, 68, 78, 202

[Estublier 05] Jacky Estublier & German Vega. *Reuse and variability in large software applications.* SIGSOFT Software Engineering Notes, vol. 30, no. 5, pages 316–325, sep 2005. 21, 22

[Fahland 09] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich & Stefan Zugal. *Declarative versus Imperative Process Modeling Languages: The Issue of Understandability.* In Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer & Roland Ukor, editors, Enterprise, Business-Process and Information Systems Modeling, volume 29 of *Lecture Notes in Business Information Processing*, pages 353–366. Springer Berlin Heidelberg, 2009. 26, 27

[Feiler 93] Peter Feiler & Watts Humphrey. *Software process development and enactment: concepts and definitions.* In Proceedings of the Second International Conference on the Software Process. Continuous Software Process Improvement, pages 28–40, 1993. 7, 11

[Foundation 10] The Yawl Foundation. *YAWL - Technical Manual, Version 2.1*, 2010. 59

[Fuggetta 00] Alfonso Fuggetta. *Software process: a roadmap.* In Proceedings of the Conference on The Future of Software Engineering, ICSE'00, pages 25–34, New York, NY, USA, 2000. ACM. 1, 3

[Giese 00] H. Giese. *Contract-based component system design.* In Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000., pages 10 pp.–, 2000. 24

[Golra 11] Fahad Rafique Golra & Fabien Dagnat. *The lazy initialization multilayered modeling framework: NIER track.* In Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011, pages 924–927, 2011. 73, 181

[Golra 12a] Fahad Rafique Golra & Fabien Dagnat. *Generation of dynamic process models for multi-metamodel applications.* In Proceedings of the International Conference on Software and System Process (ICSSP), 2012, pages 48–57, 2012. 40

[Golra 12b] Fahad Rafique Golra & Fabien Dagnat. *Specifying the Interaction Control Behavior of a Process Model using Hierarchical Petri Net.* In PMDE 2012: 2nd Workshop on Process-based approaches for Model-Driven Engineering, 2012. 176, 181

[Gonzalez-Perez 06] Cesar Gonzalez-Perez & Brian Henderson-Sellers. *A powertype-based metamodelling framework.* Software & Systems Modeling, vol. 5, no. 1, pages 72–90, 2006. 73

[Gonzalez-Perez 07] Cesar Gonzalez-Perez. *Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach.* In Jolita Ralyté, Sjaak Brinkkemper & Brian Henderson-Sellers, editors, Situational Method Engineering: Fundamentals and Experiences, volume 244 of *IFIP - The International Federation for Information Processing*, pages 7–18. Springer US, 2007. 41, 60

[Göser 07] Kevin Göser, Martin Jurisch, Hilmar Acker, Ulrich Kreher, Markus Lauer, Stefanie Rinderle-Ma, Manfred Reichert & Peter Dadam. *Next-generation Process Management with ADEPT2.* In BPM'07 Demo Proceedings, numéro 272 in CEUR-WS.org Workshop Proceedings, pages 3–6. CEUR-WS, September 2007. 36

[Hallows 02] Jolyon Hallows. The project management office toolkit. Amacom, 2002. 84

[Havey 09] Michael Havey. Essential business process modeling. O'Reilly Media, Inc., 2009. 35, 53

[Hepp 05] M. Hepp, F. Leymann, J. Domingue, A. Wahler & D. Fensel. *Semantic business process management: a vision towards using semantic Web services for business process management.* In Proceedings of the IEEE International Conference on e-Business Engineering, 2005. ICEBE 2005., pages 535–540, 2005. 29

[Hollenbach 95] Craig R. Hollenbach. Software process reusability in an industrial setting. Master's thesis, Virginia Polytechnic Institute and State University, oct 1995. 22, 23

[Hollingsworth 95] David Hollingsworth. The Workflow Reference Model, Document No. TC00-1003. The Workflow Management Coalition, 1995. 6, 29

[Hollingsworth 04] David Hollingsworth. *The Workflow Reference Model: 10 Years On.* In Fujitsu Services, UK; Technical Committee Chair of WfMC, pages 295–312, 2004. 3, 6

[Holt 83] Anatol W. Holt, R. Ramsey & J. Grimes. *Coordinating System Technology as the Basis for a Programming Environment.* Electrical Communication, vol. 57, no. 4, pages 307–314, 1983. 2

[Holt 88] Anatol W. Holt. *Diplans: a new language for the study and implementation of coordination.* ACM Transactions on Information Systems (TOIS), vol. 6, no. 2, pages 109–125, apr 1988. 2

[Hruby 98] Pavel Hruby. *Specification of workflow management systems with UML.* In OOPSLA Workshop on Implementation and Application of Object-oriented Workflow Management Systems, 1998. 2

[Huhns 05] M.N. Huhns & M.P. Singh. *Service-oriented computing: key concepts and principles.* IEEE Internet Computing, vol. 9, no. 1, pages 75–81, 2005. 27, 28, 202

[Hurtado Alegría 11] Julio A. Hurtado Alegría, María Cecilia Bastarrica, Alcides Quispe & Sergio F. Ochoa. *An MDE approach to software process tailoring.* In Proceedings of the 2011 International Conference on Software and Systems Process, ICSSP '11, pages 43–52, New York, NY, USA, 2011. ACM. 32

[Hurtado 12] Julio Ariel Hurtado, María Cecilia Bastarrica, Sergio F. Ochoa & Jocelyn Simmonds. *MDE software process lines in small companies.* Journal of Systems and Software, 2012. 10

[IEEE 06] IEEE. *IEEE Std 1074 ^{TM}-2006 - IEEE Standard for Developing a Software Project Life Cycle Process.* Standard, IEEE Computer Society, July 2006. 64, 65, 66, 72, 202

[IEEE 10] IEEE. *IEEE Std 1517 ^{TM}-2010 - IEE Standard for Information Technology - System and Software Life Cycle Processes - Reuse Processes.* Standard, IEEE Computer Society, August 2010. 65

[IEEE 13] IEEE. *ISO/IEC/IEEE 29119-2(E) IEEE Std 29119 ^{TM}-2013 - Software and systems engineering - Software testing - Part 2: Test processes.* Standard, IEEE Computer Society, February 2013. 132, 133, 203

[Indulska 09] Marta Indulska, Peter Green, Jan Recker & Michael Rosemann. *Business Process Modeling: Perceived Benefits.* In AlbertoH.F. Laender, Silvana Castano, Umeshwar Dayal, Fabio Casati & José PalazzoM. Oliveira, editors, Conceptual Modeling - ER 2009, volume 5829 of *Lecture Notes in Computer Science*, pages 458–471. Springer Berlin Heidelberg, 2009. 33

[ISO/IEC 08a] ISO/IEC. *ISO/IEC 12207:2008(E) IEEE Std 12207 ^{TM}-2008 - Systems and software engineering - Software life cycle processes.* Standard, IEEE Computer Society, February 2008. 65, 68

[ISO/IEC 08b] ISO/IEC. *ISO/IEC 15288:2008(E) IEEE Std 15288 ^{TM}-2008 - Systems and software engineering - System life cycle processes.* Standard, IEEE Computer Society, February 2008. 65

[ISO/IEC 08c] ISO/IEC. *ISO/IEC 24744:2007 - Software engineering - Metamodel for Development Methodologies.* Standard, feb 2008. 73, 181

[Jaccheri 93] M.L. Jaccheri & R. Conradi. *Techniques for process model evolution in EPOS.* IEEE Transactions on Software Engineering, vol. 19, no. 12, pages 1145–1156, 1993. 41, 60

[Johnson 99] Donna L. Johnson & Judith G. Brodman. *Tailoring the CMM for small businesses, small organizations, and small projects.* Elements of software process assessment and improvement, pages 239–259, 1999. 10

[Jones 02] M. Jones, E. Gomez, A. Mantineo & U.K. Mortensen. *Introducing ECSS Software-Engineering Standards within ESA - Practical approaches for space- and ground-segment software.* ESA Bulletin, vol. 111, pages 132–139, aug 2002. 66

[Jouault 06] Frédéric Jouault, Jean Bézivin & Ivan Kurtev. *TCS:: a DSL for the specification of textual concrete syntaxes in model engineering.* In Proceedings of the 5th international conference on Generative programming and component engineering, GPCE '06, pages 249–254, New York, USA, 2006. ACM. 2, 3

[Kabbaj 07] Mohammed Kabbaj, Redouane Lbath & Bernard Coulette. *A deviation-tolerant approach to software process evolution.* In Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, IWPSE '07, pages 75–78, New York, NY, USA, 2007. ACM. 32, 34

[Kedji 12] K.A. Kedji, Redouane Lbath, Bernard Coulette, M. Nassar, L. Baresse & F. Racaru. *Supporting collaborative development using process models: An integration-focused approach.* In International Conference on Software and System Process (ICSSP), pages 120–129, June 2012. 45

[Keen 97] Peter G.W. Keen. The process edge: creating value where it counts. Harvard Business Press, 1997. 18

[Kellner 90] M.I. Kellner, P.H. Feiler, A. Finkelstein, T. Katayama, L.J. Osterweil, M.H. Penedo & H.D. Rombach. *Software Process Modeling Example Problem.* In 'Support for the Software Process', Proceedings of the 6th International Software Process Workshop, pages 19 –29, Oct 1990. 79

[Kent 02] Stuart Kent. *Model Driven Engineering.* In Michael Butler, Luigia Petre & Kaisa Sere, editors, Integrated Formal Methods, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer Berlin Heidelberg, 2002. 1, 3

[Khalfallah 13] Malik Khalfallah, Nicolas Figay, Parisa Ghodous & Catarina-Ferreira Silva. *Cross-Organizational Business Processes Modeling Using Design-by-Contract Approach.* In Marten Sinderen, Paul Oude Luttighuis, Erwin Folmer & Steven Bosems, editors, Enterprise Interoperability, volume 144 of *Lecture Notes in Business Information Processing*, pages 77–90. Springer Berlin Heidelberg, 2013. 23

[Kloppmann 05] Matthias Kloppmann, Dieter Koenig, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt & Ivana Trickovic. *WS-BPEL extension for people–BPEL4PEOPLE.* Joint white paper, IBM and SAP, vol. 183, page 184, 2005. 42, 54, 55

[Ko 09] Ryan KL Ko, Stephen SG Lee & Eng Wah Lee. *Business process management (BPM) standards: a survey.* Business Process Management Journal, vol. 15, no. 5, pages 744–791, 2009. 53

[Koudri 10a] Ali Koudri & Joel Champeau. *MODAL: A SPEM Extension to Improve Co-design Process Models.* In Jürgen Münch, Ye Yang & Wilhelm Schäfer, editors, New Modeling Concepts for Todays Software Processes, volume 6195 of *Lecture Notes in Computer Science*, pages 248–259. Springer Berlin Heidelberg, 2010. 41, 46

[Koudri 10b] Ali Koudri, Joël Champeau, Jean-Christophe Le Lann & Vincent Leilde. *MoPCoM Methodology: Focus on Models of Computation.* In Thomas Kühne, Bran Selic, Marie-Pierre Gervais & François Terrier, editors, Modelling Foundations and Applications, volume 6138 of *Lecture Notes in Computer Science*, pages 189–200. Springer Berlin Heidelberg, 2010. 48

[Lehman 91] M.M. Lehman. *Software engineering, the software process and their support.* Software Engineering Journal, vol. 6, no. 5, pages 243–258, 1991. 18

[Li 99] Xuandong Li & Johan Lilius. *Timing Analysis of UML Sequence Diagrams.* In Robert France & Bernhard Rumpe, editors, UML'99 - The Unified Modeling Language, volume 1723 of *Lecture Notes in Computer Science*, pages 661–674. Springer Berlin Heidelberg, 1999. 2

[Lindsay 03] Ann Lindsay, Denise Downs & Ken Lunn. *Business processes-attempts to find a definition.* Information and Software Technology, vol. 45, no. 15, pages 1015 – 1019, 2003. 18

[Lonchamp 93] Jacques Lonchamp. *A structured conceptual and terminological framework for software process engineering.* In Proceedings of the Second International Conference on the Software Process, 1993. Continuous Software Process Improvement, pages 41–53, 1993. 18

[Ma 09] Zhilei Ma & F. Leymann. *BPEL Fragments for Modularized Reuse in Modeling BPEL Processes.* In Fifth International Conference on Networking and Services, 2009. ICNS '09., pages 63–68, 2009. 55

[Maciel 13] Rita Suzana Pitangueira Maciel, Ramon Araújo Gomes, Ana Patrícia Magalhaes, Bruno C. Silva & Joao Pedro B. Queiroz. *Supporting model-driven development using a process-centered software engineering environment.* Automated Software Engineering, vol. 20, no. 3, pages 427–461, 2013. 20

[Martin 91] James Martin. Rapid application development. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1991. 36

[Medvidovic 00] N. Medvidovic & R.N. Taylor. *A classification and comparison framework for software architecture description languages.* IEEE Transactions on Software Engineering, vol. 26, no. 1, pages 70–93, 2000. 24, 25, 202

[Mendling 09] Jan Mendling. *Event-Driven Process Chains (EPC).* In Metrics for Process Models, volume 6 of *Lecture Notes in Business Information Processing*, pages 17–57. Springer Berlin Heidelberg, 2009. 57

[Mendling 10] J. Mendling, H.A. Reijers & Will M.P. van der Aalst. *Seven process modeling guidelines (7PMG).* Information and Software Technology, vol. 52, no. 2, pages 127 – 136, 2010. 33

[Meyer 92a] Bertrand Meyer. *Applying 'design by contract'.* Computer, vol. 25, no. 10, pages 40–51, 1992. 22, 39, 104

[Meyer 92b] Bertrand Meyer. Eiffel: the language. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. 22, 104

[Montoni 06] Mariano Montoni, Gleison Santos, Ana Regina Rocha, Sávio Figueiredo, Reinaldo Cabral, Rafael Barcellos, Ahilton Barreto, Andréa Soares, Cristina Cerdeiral & Peter Lupo. *Taba Workstation: Supporting Software Process Deployment Based on CMMI and MR-MPS.BR.* In Jürgen Münch & Matias Vierimaa, editors, Product-Focused Software Process Improvement, volume 4034 of *Lecture Notes in Computer Science*, pages 249–262. Springer Berlin Heidelberg, 2006. 20

[Mutschler 08] B. Mutschler, M. Reichert & J. Bumiller. *Unleashing the Effectiveness of Process-Oriented Information Systems: Problem*

Analysis, Critical Success Factors, and Implications. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 38, no. 3, pages 280–291, 2008. 122

[Naur 69] Peter Naur & Brian Randell, editors. Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 oct. 1968. Scientific Affairs Division, NATO, Brussels, 1969. 1

[OASIS 07] OASIS. *Web Services Business Process Execution Language (WS-BPEL), Version 2.0*, may 2007. 2, 3, 4, 19, 29, 30, 42, 46, 52, 53, 76, 202

[OMG 08] OMG. *Software & Systems Process Engineering Metamodel Specification (SPEM), Version 2.0*, apr 2008. 3, 5, 32, 37, 38, 41, 42, 76, 84, 202

[OMG 11] OMG. *Business Process Model And Notation (BPMN), Version 2.0*, jan 2011. 2, 3, 4, 5, 19, 32, 48, 50, 52, 76, 202

[Openflexo 13] Openflexo. *Open source business architecture platform.* www.openflexo.org, 2013. 12, 118

[Osterweil 87] Leon J. Osterweil. *Software processes are software too.* In Proceedings of the 9th international conference on Software Engineering, ICSE '87, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. 1, 2, 3, 19, 24

[Ould 95] Martyn A. Ould. Business Processes: Modelling and analysis for re-engineering and improvement. Wiley, Beverly Hills, 1995. 2

[Ouyang 06] Chun. Ouyang, Marlon Dumas, Arthur H.M. ter Hofstede & Wil M.P. van der Aalst. *From BPMN Process Models to BPEL Web Services.* In Proceedings of the International Conference on Web Services, 2006. ICWS '06, pages 285–292, 2006. 32, 52

[Papazoglou 07] MikeP. Papazoglou & Willem-Jan Heuvel. *Service oriented architectures: approaches, technologies and research issues.* The VLDB Journal, vol. 16, no. 3, pages 389–415, 2007. 27, 28

[Pesić 08] Maja Pesić. *Constraint-Based Work on Management Systems: Shifting Control to Users.* PhD thesis, Eindhoven University of Technology, 2008. 26

[Pichler 12] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling & HajoA. Reijers. *Imperative versus Declarative Process Modeling Languages: An Empirical Investigation.* In Florian Daniel, Kamel Barkaoui & Schahram Dustdar, editors, Business Process Management Workshops, volume 99 of *Lecture Notes in Business Information Processing*, pages 383–394. Springer Berlin Heidelberg, 2012. 26, 27

[Pillain 11] Pierre-Yves Pillain, Joel Champeau & Hanh Nhi Tran. *Towards an Enactment Mechanism for MODAL Process Models.* In First

Workshop on Process-based approaches for Model-Driven Engineering (PMDE), 2011, page 33, June 2011. 3, 46, 48

[Portela 12] Carlos Portela, Alexandre Vasconcelos, Antônio Silva, Elder Silva, Mariano Gomes, Maurício Ronny, Wallace Lira & Sandro Oliveira. *xSPIDER_ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0.* Journal of Software Engineering and Applications, vol. 5, no. 6, pages 375–384, 2012. 41, 46

[Prieto-Diaz 87] Ruben Prieto-Diaz & Peter Freeman. *Classifying Software for Reusability.* IEEE Software, vol. 4, no. 1, pages 6–16, 1987. 9

[Prieto-Díaz 93] Rubén Prieto-Díaz. *Status report: software reusability.* Software, IEEE, vol. 10, no. 3, pages 61–66, 1993. 20

[Recker 06a] J. Recker, M. Indulska, M. Rosemann & P. Green. *How Good is BPMN Really? Insights from Theory and Practice.* In Proceedings of the 14th European Conference on Information Systems. Goeteborg, Sweden, pages 1582–1593, 2006. 48

[Recker 06b] Jan C Recker & Jan Mendling. *On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages.* In The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium, pages 521–532. Namur University Press, 2006. 54

[Rosemann 06a] Michael Rosemann. *Potential pitfalls of process modeling: part A.* Business Process Management Journal, vol. 12, no. 2, pages 249–254, 2006. 34

[Rosemann 06b] Michael Rosemann. *Potential pitfalls of process modeling: part B.* Business Process Management Journal, vol. 12, no. 3, pages 377–384, 2006. 33, 34

[Rossi 07] D. Rossi & E. Turrini. *Using a process modeling language for the design and implementation of process-driven applications.* In Proceedings of the International Conference on Software Engineering Advances. ICSEA 2007, pages 55–55, 2007. 29

[Russel 07] Nick Charles Russel. *Foundations of Process-Aware Information Systems.* PhD thesis, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, dec 2007. 58

[Russell 05a] Nick Russell, Arthur H.M. ter Hofstede, David Edmond & Wil M.P. van der Aalst. *Workflow Data Patterns: Identification, Representation and Tool Support.* In Lois Delcambre, Christian Kop, Heinrich C. Mayr, John Mylopoulos & Oscar Pastor, editors, Conceptual Modeling - ER 2005, volume 3716 of *Lecture Notes in Computer Science*, pages 353–368. Springer Berlin Heidelberg, 2005. 13, 153, 154, 155, 157, 203

[Russell 05b] Nick Russell, Wil M.P. van der Aalst, Arthur H.M. ter Hofstede & David Edmond. *Workflow resource patterns: Identification, representation and tool support.* In Advanced Information Systems Engineering, pages 216–232. Springer, 2005. 13, 153

[Russell 06a] Nick Russell, Arthur H.M. ter Hofstede, Wil M.P. van der Aalst & Nataliya Mulyar. *Workflow Control-Flow Patterns: A Revised View.* Technical Report BPM Center Report BPM-06-22, BPMcenter.org, 2006. 12, 153, 160

[Russell 06b] Nick Russell, Wil M.P. van der Aalst & Arthur H.M. ter Hofstede. *Workflow Exception Patterns.* In Eric Dubois & Klaus Pohl, editors, Advanced Information Systems Engineering, volume 4001 of *Lecture Notes in Computer Science*, pages 288–302. Springer Berlin Heidelberg, 2006. 153

[Russell 07] Nicholas Charles Russell. *Foundations of Process-Aware Information Systems.* PhD thesis, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, 2007. 154, 160

[Russell 09] Nick Russell & Arthur H.M. ter Hofstede. *Surmounting BPM challenges: the YAWL story.* Computer Science - Research and Development, vol. 23, no. 2, pages 67–79, 2009. 58, 59

[Sadiq 07] Shazia Sadiq, Guido Governatori & Kioumars Namiri. *Modeling Control Objectives for Business Process Compliance.* In Gustavo Alonso, Peter Dadam & Michael Rosemann, editors, Business Process Management, volume 4714 of *Lecture Notes in Computer Science*, pages 149–164. Springer Berlin Heidelberg, 2007. 10, 73

[Schall 08] D. Schall, Hong-Linh Truong & S. Dustdar. *Unifying Human and Software Services in Web-Scale Collaborations.* Internet Computing, IEEE, vol. 12, no. 3, pages 62–68, 2008. 42

[Scheer 00] August-Wilhelm Scheer & Markus Nüttgens. *ARIS Architecture and Reference Models for Business Process Management.* In Wil M.P. van der Aalst, Jörg Desel & Andreas Oberweis, editors, Business Process Management, volume 1806 of *Lecture Notes in Computer Science*, pages 376–389. Springer Berlin Heidelberg, 2000. 55

[Scheer 05] August-Wilhelm Scheer, Oliver Thomas & Otmar Adam. *Process modeling using event-driven process chains.* In Marlon Dumas, Wil M.P. van der Aalst & Arthur H.M. ter Hofstede, editors, Process-Aware Information Systems, pages 119–146. Wiley, Hoboken, New Jersey, 2005. 56

[Scheer 09] August-Wilhelm Scheer. Business process engineering: reference models for industrial enterprises. Springer-Verlag Telos, 2009. 56

[Selic 03] Bran Selic. *The pragmatics of model-driven development.* Software, IEEE, vol. 20, no. 5, pages 19–25, 2003. 2

[Snowdon 90] Robert A. Snowdon. *An Introduction to the IPSE 2.5 project.* In Fred Long, editor, Software Engineering Environments, volume 467 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin Heidelberg, 1990. 2

[Society 10] IEEE Computer Society. *IEEE Standard for Information Technology - System and Software Life Cycle Processes – Reuse Processes.* IEEE Std. 1517-2010, pages i –38, 2010. 21

[Sutton Jr. 97] Stanley M Sutton Jr. & Leon J. Osterweil. *The design of a next-generation process language.* In Mehdi Jazayeri & Helmut Schauer, editors, Software Engineering – ESEC/FSE'97, volume 1301 of *Lecture Notes in Computer Science*, pages 142–158. Springer Berlin Heidelberg, 1997. 20, 33, 63

[Szyperski 97] Clemens Szyperski. Component software, Beyond object-oriented programming. Addison-Wesley, 1997. 24

[Tan 09] Wei Tan, Yushun Fan, MengChu Zhou & MengChu Zhou. *A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language.* IEEE Transactions on Automation Science and Engineering, vol. 6, no. 1, pages 94–106, 2009. 30

[Taylor 88] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon J. Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf & Michael Young. *Foundations for the Arcadia environment architecture.* ACM SIGSOFT Software Engineering Notes, vol. 13, no. 5, pages 1–13, nov 1988. 20

[Team 10] CMMI Product Team. *Improving processes for developing better products and services, CMMI ® for Development, Version 1.3.* Technical report, Carnegie Mellon University, November 2010. 26

[Thu 05] TRAN Dan Thu, TRAN Hanh Nhi, Dong Thi Bich Thuy, Bernard Coulette & Xavier Cregut. *Topological properties for characterizing well-formedness of process components.* Software Process: Improvement and Practice, vol. 10, no. 2, pages 217–247, 2005. 72

[Tran 07] Hanh Nhi Tran, Bernard Coulette & Dong Thi Bich Thuy. *Broadening the Use of Process Patterns for Modeling Processes.* In International Conference on Software Engineering and Knowledge Engineering, pages 57–62, 2007. 72

[Tran 11] Hanh Nhi Tran, Bernard Coulette, Dan Thu Tran & My Hang Vu. *Automatic reuse of process patterns in process modeling.* In Proceedings of the 2011 ACM Symposium on Applied Computing, pages 1431–1438. ACM, 2011. 72

[Turan 12] Yenal Turan. Extension and Application of Event- driven Process Chain for Information System Security Risk Management. Master's thesis, University of Tartu, may 2012. 56, 202

[Türetken 07] Oktey Türetken. *A method for decentralized business process modeling.* PhD thesis, The Middle East Technical University, 2007. 20

[Valetto 01] Giuseppe Valetto, Gail E. Kaiser & Gaurav S. Kc. *A Mobile Agent Approach to Process-Based Dynamic Adaptation of Complex Software Systems.* In Proceedings of the 8th European Workshop on Software Process Technology, EWSPT '01, pages 102–116, London, UK, UK, 2001. Springer-Verlag. 36

[van der Aalst 97] Wil M.P. van der Aalst. *Verification of workflow nets.* In Pierre Azéma & Gianfranco Balbo, editors, Application and Theory of Petri Nets 1997, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer Berlin Heidelberg, 1997. 5

[van der Aalst 99] Will M.P. van der Aalst. *Formalization and verification of event-driven process chains.* Information and Software Technology, vol. 41, no. 10, pages 639 – 650, 1999. 3

[van der Aalst 00] Wil M.P. van der Aalst & Stefan Jablonski. *Dealing with workflow change: identification of issues and solutions.* Computer systems science and engineering, vol. 15, no. 5, pages 267–276, 2000. 122

[van der Aalst 03a] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, B. Kiepuszewski & A.P. Barros. *Workflow Patterns.* Distributed and Parallel Databases, vol. 14, no. 1, pages 5–51, 2003. 153, 160

[van der Aalst 03b] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, Bartek Kiepuszewski & Alistair P. Barros. *Workflow patterns.* Distributed and parallel databases, vol. 14, no. 1, pages 5–51, 2003. 12

[van der Aalst 04] Wil M.P. van der Aalst & Kees Max van Hee. Workflow management: models, methods, and systems. The MIT press, 2004. 58, 76

[van der Aalst 05a] Wil M.P. van der Aalst & Ana Karla A de Medeiros. *Process mining and security: Detecting anomalous process executions and checking process conformance.* Electronic Notes in Theoretical Computer Science, vol. 121, pages 3–21, 2005. 73

[van der Aalst 05b] Wil M.P. van der Aalst, Marlon Dumas, Arthur H.M. ter Hofstede, Nick Russell, H.M.W. Verbeek & P. Wohed. *Life After BPEL?* In Mario Bravetti, Leïla Kloul & Gianluigi Zavattaro, editors, Formal Techniques for Computer Systems and Business Processes, volume 3670 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin Heidelberg, 2005. 54

[van der Aalst 05c] Wil M.P. van der Aalst & Arthur H.M. ter Hofstede. *YAWL: yet another workflow language.* Information Systems, vol. 30, no. 4, pages 245 – 275, 2005. 5

[van der Aalst 07] Wil M.P. van der Aalst, Hajo A Reijers, Anton JMM Weijters, Boudewijn F van Dongen, AK Alves de Medeiros, Minseok Song & HMW Verbeek. *Business process mining: An industrial application.* Information Systems, vol. 32, no. 5, pages 713–732, 2007. 3, 71

[van der Aalst 12] Wil M.P van der Aalst & Arthur H.M. ter Hofstede. *Workflow patterns put into context.* Software & Systems Modeling, vol. 11, no. 3, pages 319–323, 2012. 59

[Vanderfeesten 08] Irene Vanderfeesten, Hajo A. Reijers & Wil M.P. van der Aalst. *Evaluating workflow process designs using cohesion and coupling metrics.* Computers in Industry, vol. 59, no. 5, pages 420 – 437, 2008. 24

[Weigold 10] Thomas Weigold. *A generic framework for process execution and secure multiparty transaction authorization.* PhD thesis, University of Westminster, 2010. 31, 202

[Weigold 12] Thomas Weigold, Marco Aldinucci, Marco Danelutto & Vladimir Getov. *Process-driven biometric identification by means of autonomic grid components.* International Journal of Autonomous and Adaptive Communications Systems, vol. 5, no. 3, pages 274–291, 2012. 19

[White 08] Stephen A. White & Derek Miers. BPMN modeling and reference guide: understanding and using BPMN. Future Strategies Inc., 2008. 48

[Whittle 96] Ben Whittle, Wing Lam & Tim Kelly. *A Pragmatic Approach to Reuse Introduction in an Industrial Setting.* In Marjan Sarshar, editor, Systematic Reuse: Issues in Initiating and Improving a Reuse Program, pages 104–115. Springer London, 1996. 21

[Wise 98] Alexander Wise. *Little-JIL 1.0 language report.* Technical report, Department of Computer Science, University of Massachusetts at Amherst, 1998. 63

[Wise 11] Alexander Wise, Aaron G. Cass, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil & Stanley M. Sutton Jr. *Using Little-JIL to Coordinate Agents in Software Engineering.* In Peri L. Tarr & Alexander L. Wolf, editors, Engineering of Software, pages 383–397. Springer Berlin Heidelberg, 2011. 61

List of Figures

List of Tables

Appendices

Appendix A

Further discussion on individual Workflow Patterns

Precisions for pattern implementations

- The terms parent and child activities are used in this text as form of containment descriptors. The parent activities are the activities that compose the process containing child activities.
- Tasks used in the workflow patterns are same as primitive activities in CPMF.
- Data-elements in the work patterns are same as artifacts in CPMF.

1.1 Data-flow Patterns

Pattern 1 (Task Data)

Description: *Data elements can be defined by tasks which are accessible only within the context of individual execution instances of that task.*

Implementation: Data elements are defined in CPMF process models in the activity definitions at the abstract level. These data elements need to have a unique name within the scope of the activity. These data elements are not accessible outside the execution context of individual activity, unless provided by the contracts of the activity. Each activity definition can be implemented by multiple instantiation activities. During execution, these data elements get their 'value' at the concrete level of the process model. Each instantiation activity can have multiple instances during execution. The data element of an activity instance is accessible only within the scope of that instance.

Evaluated rating: *Direct support (+)*

Pattern 2 (Block Data)

Description: *Block tasks (i.e. tasks which can be described in terms of a corresponding sub-workflow) are able to define data elements which are accessible by each of the components of the corresponding sub-workflow.*

Implementation: Activity definitions in CPMF process model can be implemented through composite instantiation activities. A composite instantiation activity contains other instantiation activities that have access to the data elements of their parent activities. A delegation contract in activities allows to pass on the required data elements to the parent activity to its contained activities. Contained activities need to specify their required contracts in order to access the data elements offered by the parent activities. Parent activities have the choice of sharing/hiding data elements from their child activities through the use of contracts.

Evaluated rating: *Direct support (+)*

Pattern 3 (Scope Data)

Description: *Data elements can be defined which are accessible by a subset of the tasks in a case.*

Implementation: All the interaction between the activities (even the parent activities and their child activities) is handled through the use of contracts. A data dependency between two activities is specified through *binding* at the abstract level. Event-based concrete contracts of the activities conform to the abstract contracts, during execution. A composite activity can be implemented through a set of child activities. If required contracts for a particular data element are defined for a subset of these child activities, they would be able to access this data element. Other sub-activities that do not defined the required contract for this data element can not access it. Thus the scope of data element access within a composite activity can be defined through the required contracts of the contained activities.

Evaluated rating: *Direct support (+)*

Pattern 4 (Multiple Instance Data)

Description: *Tasks which are able to execute multiple times within a single workflow case can define data elements which are specific to an individual execution instance.*

Implementation: Multiple instances of an instantiation activity are possible in CPMF. In this case, they all will have the same contracts as defined in the instantiation activity. The transfer of data elements between two activities in CPMF is implemented through events. An artifact repository is a shared data store between the activities. Once the event associated with the data element is received by an activity instance, it can access the data element from the shared artifact repository. Each activity instance in CPMF framework then creates its own working copy of the element. For the artifacts that are physical and not copyable can be accessed by only one activity instance at a time. An artifact object for such data elements keeps track of the current possessor of the artifact.

Evaluated rating: *Direct support (+)*

Pattern 5 (Case Data)

Description: *Data elements are supported which are specific to a process instance or case of a workflow. They can be accessed by all components of the workflow during the execution of the case.*

Implementation: Data elements are encapsulated in the activities and are only shared through the specified contracts. CPMF does not offer a direct support for sharing the data elements of an activity to all the components of the workflow, without a well specified binding between the contracts at the abstract level. However a partial support for such situations is possible through the use of a mutually shared artifact repository and the event propagation mechanism in CPMF. Artifact events related to case data elements are broadcasted to all the contained activities of the current instance resulting in a similar effect. In this case, all activities who want to access the data elements need to specify the required contract. CPMF strongly supports the idea of contract specification for all interactions between the activities.

Evaluated rating: *Partial support (+/-)*

Pattern 6 (Folder Data)

Description: *Data elements can be defined which are accessible by multiple cases on a selective basis.*

Implementation: The artifact repository used in CPMF tool support supports concurrent version management. This helps in managing access to the data elements. But this does not allow selective accessibility of data elements for different cases.

Evaluated rating: *Support missing (-)*

Pattern 7 (Workflow Data)

Description: *Data elements are supported which are accessible to all components in each and every case of the workflow and are within the control of the workflow system.*

Implementation: Sharing a data element with all the instances of all the elements of the process model is possible because of the use of a common artifact repository. However, all the activities that need to access this data element have to specify their required contract for this data element. Events related to such data elements are broadcasted to all the activities of the process model. The concurrent access of such data element is handled through the use of concurrent version management system that allows to manage the access rights through a locking mechanism.

Evaluated rating: *Direct support (+)*

Pattern 8 (Environment Data)

Description: *Data elements which exist in the external operating environment are able to be accessed by components of the workflow during execution.*

Implementation: Activities in CPMF framework need to specify contracts for interacting with each other. However in order to interact with the environment, no contracts are required. This allows activities to interact with the development environment and invoke other development tools. Data from the environment can also be accessed by the activities without the need of any specific contract.

Evaluated rating: *Direct support (+)*

Pattern 9 (Data Interaction - Task to task)

Description: *The ability to communicate data elements between one task instance and another within the same case.*

Implementation: Two activity instances need to define their required and provided contracts in order to interact with each other. An explicit binding between the two contracts is specified at the abstract level activity definitions. However activity instances do not specify any dependency to any other activity.

Evaluated rating: *Direct support (+)*

Pattern 10 (Data Interaction - Block Task to Sub-Workflow Decomposition)

Description: *The ability to pass data elements from a block task instance to the corresponding sub-workflow that defines its implementation.*

Implementation: A block task in CPMF is represented through a composite activity. A composite activity instance contains a process composed of other activity instances. All the contracts of composite activities in CPMF, are defined in pairs. Each external contract has a corresponding internal contract of opposite direction. Thus a required external contract has a corresponding provided internal contract. Thus an activity can pass a data element to the sub-activities, even deep in the hierarchy.

Evaluated rating: *Direct support (+)*

Pattern 11 (Data Interaction - Sub-Workflow Decomposition to Block Task)

Description: *The ability to pass data elements from the underlying sub-workflow back to the corresponding block task instance.*

Implementation: The contracts of the composite activity defined in pair are valid for both internal and external contracts. Thus each internal contract of a composite activity has a corresponding external contract with opposite direction *i.e.* a required internal contract will have a paired provided external contract. This allows for data interactions initiating from child activities to their parent activities, and even higher in the hierarchy.

Evaluated rating: *Direct support (+)*

Pattern 12 (Data Interaction - to Multiple Instance Task)

Description: *The ability to pass data elements from a preceding task instance to a subsequent task which is able to support multiple execution instances. This may involve passing the data elements to all instances of the multiple instance task or distributing them on a selective basis.*

Implementation: Multiple instances of an instantiation activity are allowed in CPMF process. Data interaction between a preceding activity to multiple instances of the subsequent activities is also possible if a binding exists between the activity definitions of the subsequent activity and that of the multiple instance activity. Because the data element is kept in the artifact repository, concurrent access to this artifact does not pose any issue for 'read' access. Every activity instance has a separate working copy of the data element. 'Write' access to repository depends on the specific case, where branching or merging is allowed by the concurrent version management system.

Evaluated rating: *Direct support (+)*

Pattern 13 (Data Interaction - from Multiple Instance Task)

Description: *The ability to pass data elements from a task which supports multiple execution instances to a subsequent task.*

Implementation: Multiple instances of an instantiation activity can interact with a subsequent activity through the use of common artifact repository. Events triggered by each instance are received at the required concrete contract of the subsequent activity. Once the concerned events are received, this activity can access the data element from the repository. The use of pre-conditions allows for selective interactions as well.

Evaluated rating: *Direct support (+)*

Pattern 14 (Data Interaction - Case to Case)

Description: *The passing of data elements from one case of a workflow during its execution to another case that is executing concurrently.*

Implementation: A *Case* in CPMF corresponds to the instance of a concrete process. A concrete process instance is contained by the instance of a composite instance activity. Two instances of the same instantiation activity can only interact, if the activity definition of the instantiation activity specifies a corresponding required contract for its own provided contract. However no synchronizations are offered by CPMF for such situations. A basic synchronization can be achieved by exploiting the pre-conditions of the activities.

Evaluated rating: *Partial support (+/-)*

Pattern 15 (Data Interaction - Task to Environment - Push-Oriented))

Description: *The ability of a task to initiate the passing of data elements to a resource or service in the operating environment.*

Implementation: Data interactions of CPMF activities with its environment does not require the specification of contracts. Any activity instance can interact with its environment to pass on the data element. The interaction concerning data elements between activities uses a common repository. However, when interacting with the environment, data element can be passed without using the repository. Roles are associated with activities, which can be played by tools. Environment interactions can be handled through the associated tools.

Evaluated rating: *Partial support (+/-)*

Pattern 16 (Data Interaction - Environment to Task - Pull-Oriented)

Description: *The ability of a workflow task to request data elements from resources or services in the operational environment.*

Implementation: Requesting a data element from the activity instance uses the same mechanism as passing the data to the environment. The data element does not need to be passed through the artifact repository. Interactions with the environment are highly dependent on the individual implementations of each activity. No constraints are imposed from the CPMF framework. Roles are associated with activities, which can be played by tools. Environment interactions can be handled through the associated tools.

Evaluated rating: *Partial support (+/-)*

Pattern 17 (Data Interaction - Environment to Task - Push-Oriented)

Description: *The ability for a workflow task to receive and utilize data elements passed to it from services and resources in the operating environment on an unscheduled basis.*

Implementation: The data interaction between the environment and process elements does not follow the same structure as the data interactions between the activity instances. CPMF activities are allowed to interact with the environment. Interactions between an activity and some application in its environment to receive data elements is possible based on the implementation of the activity. Roles are associated with activities, which can be played by tools. Environment interactions can be handled through the associated tools.

Evaluated rating: *Partial support (+/-)*

Pattern 18 (Data Interaction - Task to Environment - Pull-Oriented)

Description: *The ability of a workflow task to receive and respond to requests for data elements from services and resources in the operational environment.*

Implementation: Interactions with the environment are highly dependent on the individual implementations of each activity. No constraints are imposed from the CPMF framework. An activity can request any data element from its environment, without the need of specifying a contract. Roles are associated with activities, which can be played by tools. Environment interactions can be handled through the associated tools.

Evaluated rating: *Partial support (+/-)*

Pattern 19 (Data Interaction - Case to Environment - Push-Oriented)

Description: *The ability of a workflow case to initiate the passing of data elements to a resource or service in the operational environment.*

Implementation: A *Case* is an instance of a software process. Every process in CPMF is contained in a container activity. A container activity can not contain multiple processes. Thus all interaction to/from a process to the environment take place through the container activity. A *Case* in CPMF represents an instance of a composite activity. Thus this pattern is handled the same way as pattern 15.

Evaluated rating: *Partial support (+/-)*

Pattern 20 (Data Interaction - Environment to Case - Pull-Oriented)

Description: *The ability of a workflow case to request data from services or resources in the operational environment.*

Implementation: This pattern is handled the same way as pattern 16, as *Case* in CPMF corresponds to a composite activity instance.

Evaluated rating: *Partial support (+/-)*

Pattern 21 (Data Interaction - Environment to Case - Push-Oriented)

Description: *The ability of a workflow case to accept data elements passed to it from services or resources in the operating environment.*

Implementation: This pattern is handled the same way as pattern 17, as *Case* in CPMF corresponds to a composite activity instance.

Evaluated rating: *Partial support (+/-)*

Pattern 22 (Data Interaction - Case to Environment - Pull-Oriented)

Description: *Data elements can be defined by tasks which are accessible only within the context of individual execution instances of that task.*

Implementation: This pattern is handled the same way as pattern 18, as *Case* in CPMF corresponds to a composite activity instance.

Evaluated rating: *Partial support (+/-)*

Pattern 23 (Data Interaction - Workflow to Environment - Push-Oriented)

Description: *The ability of a workflow engine to pass data elements to resources or services in the operational environment.*

Implementation: CPMF processes are not executed on a workflow engine. An interpreter serves as a process engine and is used to execute these processes. An artifact repository is directly associated with the interpreter. All interactions between the environment and the interpreter do not need to pass through the artifact repository. However, the interpreter can access the data elements from the repository to pass on to the environment. A process in CPMF is always contained by a container activity. Thus a root activity contains all the processes of a process model. The interaction of the environment with the process model is in fact its interaction with the root activity. Roles are associated with activities, which can be played by tools. Environment interactions can be handled through the associated tools.

Evaluated rating: *Partial support (+/-)*

Pattern 24 (Data Interaction - Environment to Workflow - Pull-Oriented)

Description: *The ability of a workflow to request workflow-level data elements from external applications.*

Implementation: Please refer to the implementation of Pattern 23 for the implementation details.

Evaluated rating: *Partial support (+/-)*

Pattern 25 (Data Interaction - Environment to Workflow - Push-Oriented)

Description: *The ability of services or resources in the operating environment to pass workflow-level data to a workflow process.*

Implementation: Please refer to the implementation of Pattern 23 for the implementation details.

Evaluated rating: *Partial support (+/-)*

Pattern 26 (Data Interaction - Workflow to Environment - Pull-Oriented)

Description: *The ability of a workflow engine to handle requests for workflow-level data from external applications.*

Implementation: Please refer to the implementation of Pattern 23 for the implementation details.

Evaluated rating: *Partial support (+/-)*

Pattern 27 (Data Transfer by Value - Incoming)

Description: *The ability of a workflow component to receive incoming data elements by value relieving it from the need to have shared names or common address space with the component(s) from which it receives them.*

Implementation: The data elements are passed between the interacting activities through the reference of repository location.

Evaluated rating: *Support missing (-)*

Pattern 28 (Data Transfer by Value - Outgoing)

Description: *The ability of a workflow component to pass data elements to subsequent components as values relieving it from the need to have shared names or common address space with the component(s) to which it is passing them.*

Implementation: The data elements are passed between the interacting activities through the reference of repository location.

Evaluated rating: *Support missing (-)*

Pattern 29 (Data Transfer - Copy In/Copy Out)

Description: *The ability of a workflow component to copy the values of a set of data elements into its address space at the commencement of execution and to copy their final values back at completion.*

Implementation: A common artifact repository is associated with the process interpreter. All interactions between the activities for passing data elements is handled through the reference of repository locations of the data elements. An activity accesses the data element from the repository and creates its own working copy. Once the activity is complete it writes back the data element to the repository as a new version.

Evaluated rating: *Direct support (+)*

Pattern 30 (Data Transfer by Reference - Unlocked)

Description: *The ability to communicate data elements between workflow components by utilizing a reference to the location of the data element in some mutually accessible location. No concurrency restrictions apply to the shared data element.*

Implementation: A common artifact repository is associated with the process interpreter. All interactions between the activities for passing data elements is handled through the reference of repository locations of the data elements. An activity triggers an associated event to the data element, when it is ready to be shared. Subsequent activity can retrieve the data element from the repository.

Evaluated rating: *Direct support (+)*

Pattern 31 (Data Transfer by Reference - With Lock)

Description: *The ability to communicate data elements between workflow components by passing a reference to the location of the data element in some mutually accessible location. Concurrency restrictions are implied with the receiving component receiving the privilege of read-only or dedicated access to the data element.*

Implementation: A common artifact repository is associated with the process interpreter. All interactions between the activities for passing data elements is handled through the reference of repository locations of the data elements. An activity triggers an associated event to the data element, when it is ready to be shared. Subsequent activity can retrieve the data element from the repository. The artifact repository in CPMF supports concurrent version management. Its implementation allows a locking mechanism to the activities to manage the access rights for other activities.

Evaluated rating: *Direct support (+)*

Pattern 32 (Data Transformation - Input)

Description: *The ability to apply a transformation function to a data element prior to it being passed to a workflow component.*

Implementation: No action on data elements can be performed outside the activities. Transformation of a data element itself is considered as an activity in CPMF framework. It is not possible to apply a transformation function to a data element after it is delivered by an activity and just before it is received by some other activity.

Evaluated rating: *Support missing (-)*

Pattern 33 (Data Transformation - Output)

Description: *The ability to apply a transformation function to a data element immediately prior to it being passed out of a workflow component.*

Implementation: A transformation of a data element can be performed within an activity, as its own implementation. A composite activity can contain a transformation activity as the last executing activity before its completion. This allows to apply the transformation to the data element immediately before delivering it to other activities through provided contract.

Evaluated rating: *Direct support (+)*

Pattern 34 (Task Precondition - Data Existence)

Description: *Data-based preconditions can be specified for tasks based on the presence of data elements at the time of execution.*

Implementation: Pre-conditions in CPMF framework can validate the presence or absence of a particular data element.

Evaluated rating: *Direct support (+)*

Pattern 35 (Task Precondition - Data Value)

Description: *Data-based preconditions can be specified for tasks based on the value of specific parameters at the time of execution.*

Implementation: Pre-conditions are part of every activity in CPMF framework. They can be based on the value of the properties associated to the activities or data elements. Data elements are structured in CPMF. Each artifact maps to an artifact specification that has an associated artifact metamodel. Every artifact conforms to its metamodel. The pre-conditions of an activity can be used to validate the correctness of required artifacts.

Evaluated rating: *Direct support (+)*

Pattern 36 (Task Postcondition - Data Existence)

Description: *Data-based postconditions can be specified for tasks based on the existence of specific parameters at the time of execution.*

Implementation: Activities in CPMF also define post-conditions for the activities. These post-conditions can check the availability/absence of data elements.

Evaluated rating: *Direct support (+)*

Pattern 37 (Task Postcondition - Data Value)

Description: *Data-based postconditions can be specified for tasks based on the value of specific parameters at the time of execution.*

Implementation: Post-conditions associated with the CPMF activities can be used to validate the value of specific data elements. Data elements are structured in CPMF. Post-conditions can also validate the correctness of provided artifacts.

Evaluated rating: *Direct support (+)*

Pattern 38 (Event-based Task Trigger)

Description: *The ability for an external event to initiate a task.*

Implementation: The concrete level of CPMF framework defines the control flow of activities. The contracts of each activity contain events. Output events of an activity triggered from the provided contract. The required contract of an activity contains the event listeners. These events listeners are used to receive the events and can trigger the activities.

Evaluated rating: *Direct support (+)*

Pattern 39 (Data-based Task Trigger)

Description: *The ability to trigger a specific task when an expression based on workflow data elements evaluates to true.*

Implementation: There is no direct support for monitoring conditions based on data elements. A data-based trigger is implemented through the use of events that map to data elements. Each artifact in CPMF has an associated state machine. A change of state for an artifact triggers an event that can be received by the activity. These events based on change of artifact state along with the pre-conditions of the activities can monitor the data elements for triggering the activity.

Evaluated rating: *Partial support (+/-)*

Pattern 40 (Data-based Routing)

Description: *The ability to alter the control flow within a workflow case as a consequence of the value of data-based expressions.*

Implementation: The concrete level of CPMF framework does not provide any logical connectors for altering the control flow of the process. The logic for control flow is embedded in the contracts of the activities and the event broker presented by the container activity. In order to achieve a data-based routing in a process, this logic has to be encoded in the activity through its preconditions and the event broker provided by its container activity to manage the context.

Evaluated rating: *Partial support (+/-)*

1.2 Control-flow Patterns

Pattern 1 (Sequence)

Description: *A task in a process in enabled after the completion of a preceding task in the same process.*

Implementation: Activities to be executed in a series one after another are bound together though the contracts. This binding is possible if the provided contract of the first activity, provides the artifact, which is required by the subsequent activity. In this case both the activities are considered bound and the production of artifact from first activity might trigger the execution of the subsequent activity (if other pre-conditions are satisfied).

Evaluated rating: *Direct support (+)*

Pattern 2 (Parallel Split)

Description: *The divergence of a branch into two or more parallel branches each of which execute concurrently.*

Implementation: If two activities need to be executed in parallel after the current activity, this control flow is usually handled through an AND-Split connector. Logical connectors in CPMF are not explicitly stated. They are encoded within the contracts. No special connector is required to express a parallel split control flow. The default control-flow of two activities following a single activity is of parallel split. They both need to be bound to the same precedent activity.

Evaluated rating: *Direct support (+)*

Pattern 3 (Synchronization)

Description: *The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.*

Implementation: This control flow structure is normally handled through an AND-join connector. In CPMF, such a control flow structure can be achieved when a single activity is bound to two precedent activities. In this case, the required contract of the subsequent activity specifies the behavior (i.e. AND behavior). In this case, the pre-condition of the activity is used to define this behavior.

Evaluated rating: *Direct support (+)*

Pattern 4 (Exclusive Choice)

Description: *The divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely* *one of the outgoing branches based on a mechanism that can select one of the outgoing branches.*

Implementation: Two or more activities are followed by a single activity, where it is bound to the subsequent activities. In this case, the exclusive-OR behavior is encoded in the required contract of each subsequent activity, through the pre-conditions. The pre-conditions are used to verify the log maintained by the event broker of the parent activity. If no subsequent activity has accepted the input, then the current activity can accept this input after updating the log maintained by the event broker. This ensures that only one subsequent activity can accept the input.

Evaluated rating: *Direct support (+)*

Pattern 5 (Simple Merge)

Description: *The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch.*

Implementation: A single activity follows two or more activities, such that the precedent activities are bound to that single activity. In this case, this behavior is encoded into the required contract of the subsequent activity. The log kept by the event broker ensures the enablement of the subsequent activity. When an input is received, the activity logs the enablement and continues its execution. This ensures that its is enabled only once for a single precedent activity.

Evaluated rating: *Direct support (+)*

Pattern 6 (Multi-Choice)

Description: *The divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to one or more of the outgoing branches based on a mechanism that selects one or more outgoing branches.*

Implementation: This control-flow structure is normally handled by OR-Split connectors. In CPMF, when multiple activities follow a single activity, such a behavior can be encoded in the pattern. This behavior is encoded in the pre-conditions of the subsequent activities. These pre-conditions have access to the log maintained by the event broker. The subsequent activity listens the event and checks the log. If no other activity has logged its enablement, then it accepts the input. In case, the log already contains an enablement, then the current activity can choose to accept or reject the input non-deterministically.

Evaluated rating: *Direct support (+)*

Pattern 7 (Structured Synchronizing Merge)

Description: *The convergence of two or more branches (which diverged earlier in the process at a uniquely identifiable point) into a single subsequent branch such that the thread of control is passed to the subsequent branch when each active incoming branch has been enabled. The Structured Synchronizing Merge occurs in a structured context, i.e. there must be a single Multi-Choice construct earlier in the process model with which the Structured Synchronizing Merge is associated and it must merge all of the branches emanating from the Multi-Choice. These branches must either flow from the Structured Synchronizing Merge without any splits or joins or they must be structured in form (i.e. balanced splits and joins).*

Implementation: This control flow structure is normally handled through an OR-join connector. In CPMF, such a control flow structure can be achieved when a single activity is bound to two or more precedent activities. In this case, the required contract of the subsequent activity specifies the behavior (i.e. OR behavior) through a pre-condition. Events triggered by each of the precedent activities bound to the single subsequent activity are listened. If an event of enablement is received the subsequent activity can start its execution.

Evaluated rating: *Direct support (+)*

Pattern 8 (Multi-Merge)

Description: *The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch.*

Implementation: Such a control flow structure can be achieved when a single activity is bound to two or more precedent activities. In this case, the required contract of the subsequent activity specifies the behavior through a pre-condition and a local log for the queued events that it receives. Events triggered by each of the precedent activities bound to the single subsequent activity are listened. Whenever an event of enablement is received the subsequent activity starts its execution. If an event is received during the execution of the activity, it augments the iteration of the current activity and logs the input for the next iteration.

Evaluated rating: *Direct support (+)*

Pattern 9 (Structured Discriminator)

Description: *The convergence of two or more branches into a single subsequent branch following a corresponding divergence earlier in the process model such that the thread of control is passed to the subsequent branch when the first incoming branch has been enabled. Subsequent enablements of incoming branches do not result in the thread of control being passed on. The Structured Discriminator construct resets when all incoming branches have been enabled. The Structured Discriminator occurs in a* *structured context, i.e. there must be a single Parallel Split construct earlier in the process model with which the Structured Discriminator is associated and it must merge all of the branches emanating from the Structured Discriminator. These branches must either flow from the Parallel Split to the Structured Discriminator without any splits or joins or they must be structured in form (i.e. balanced splits and joins).*

Implementation: Such a control flow structure can be achieved when a single activity is bound to two or more precedent activities. In this case, the required contract of the subsequent activity specifies the behavior through a pre-condition and a local log for the queued events that it receives. Events triggered by each of the precedent activities bound to the single subsequent activity are listened. When an event of enablement is received the subsequent activity logs it and starts its execution. If an event is received during the execution of the activity or after its execution, it logs the events, but does not start the execution of the activity. Once events are received from all the precedent activities, the local log is cleared.

Evaluated rating: *Direct support (+)*

Pattern 10 (Arbitrary Cycles)

Description: *The ability to represent cycles in a process model that have more than one entry or exit point. It must be possible for individual entry and exit points to be associated with distinct branches.*

Implementation: Each CPMF activity allows for multiple input and output contracts. It also supports different kinds of control flow behaviors to allow arbitrary cycles in the execution of the process model.

Evaluated rating: *Direct support (+)*

Pattern 11 (Implicit Termination)

Description: *A given process (or sub-process) instance should terminate when there are no remaining work items that are able to be done either now or at any time in the future and the process instance is not in deadlock. There is an objective means of determining that the process instance has successfully completed.*

Implementation: CPMF process model allows to define a customizable life-cycle for each activity. In case a user does not define a life-cycle, a default life-cycle is followed by the activity. This life-cycle allows to terminate the activity from one or more states.

Evaluated rating: *Direct support (+)*

Pattern 12 (Multiple Instances without Synchronization)

Description: *Within a given process instance, multiple instances of a task can be created. These instances are independent of each other and run concurrently. There* *is no requirement to synchronize them upon completion. Each of the instances of the multiple instance task that are created must execute within the context of the process instance from which they were started (i.e. they must share the same case identifier and have access to the same data elements) and each of them must execute independently from and without reference to the task that started them.*

Implementation: Multiple instances of an activity are allowed in CPMF. Each of these instances have their own properties. The inputs and outputs of an activity are handled through references to the artifact repository. Each instance of an activity can refer to a unique artifact in the repository (and can also refer to different artifacts). This allows concurrent execution of each activity instance, where they are not synchronized upon completion. Each activity instance follows its own lifecycle, separate from other activity instances.

Evaluated rating: *Direct support (+)*

Pattern 13 (Multiple Instances with a *priori* Design-Time Knowledge)

Description: *Within a given process instance, multiple instances of a task can be created. The required number of instances is known at design time. These instances are independent of each other and run concurrently. It is necessary to synchronize the task instances at completion before any subsequent tasks can be triggered.*

Implementation: Multiple instances of an activity are allowed in CPMF. Each of these instances have their own properties. Design-time activity implementation allows to specify the allowed number of instances for each implementation. One can also specify the synchronization of each instance upon completion using the post-conditions of the activity implementations. Each activity instance follows its own lifecycle, separate from other activity instances. This allows concurrent execution of each activity instance, where they are synchronized upon completion.

Evaluated rating: *Direct support (+)*

Pattern 14 (Multiple Instances with a *priori* Run-Time Knowledge)

Description: *Within a given process instance, multiple instances of a task can be created. The required number of instances may depend on a number of runtime factors, including state data, resource availability and inter-process communications, but is known before the task instances must be created. Once initiated, these instances are independent of each other and run concurrently. It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered.*

Implementation: CPMF allows multiple instances of an activity in its process models. Each of these instances have their own properties. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Although there is no direct way to precise the synchronization upon completion, one can specify the synchronization of each instance upon completion using the pre-conditions of the subsequent activity implementations. Each activity instance follows its own lifecycle, separate from other activity instances. This allows concurrent execution of each activity instance, where they are synchronized upon completion.

Evaluated rating: *Partial support (+/-)*

Pattern 15 (Multiple Instances without a *priori* Run-Time Knowledge)

Description: *Within a given process instance, multiple instances of a task can be created. The required number of instances may depend on a number of runtime factors, including state data, resource availability and inter-process communications and is not known until the final instance has completed. Once initiated, these instances are independent of each other and run concurrently. At any time, whilst instances are running, it is possible for additional instances to be initiated. It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered.*

Implementation: CPMF allows multiple instances of an activity in its process models. Each of these instances have their own properties. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Runtime adaptation to the activities allows to update this property, which can be based on any runtime factor. Although there is no direct way to precise the synchronization upon completion, one can specify the synchronization of each instance upon completion using the pre-conditions of the subsequent activity implementations. Each activity instance follows its own lifecycle, separate from other activity instances. This allows concurrent execution of each activity instance, where they are synchronized upon completion.

Evaluated rating: *Partial support (+/-)*

Pattern 16 (Deferred Choice)

Description: *A point in a process where one of several branches is chosen based on interaction with the operating environment. Prior to the decision, all branches represent possible future courses of execution. The decision is made by initiating the first task in one of the branches i.e. there is no explicit choice but rather a race between different branches. After the decision is made, execution alternatives in branches other than the one selected are withdrawn.*

Implementation: Two or more activities are followed by a single activity, where it is bound to the subsequent activities. In this case, the exclusive-OR behavior is encoded in the required contract of each subsequent activity, through the pre-conditions. The pre-conditions are used to verify the log maintained by the event broker of the parent activity. If no subsequent activity has accepted the input, then the current activity can accept this input after updating the log maintained by the event broker. This induces a race between different branches and ensures that only the first activity can accept the input.

Evaluated rating: *Direct support (+)*

Pattern 17 (Interleaved Parallel Routing)

Description: *A set of tasks has a partial ordering defining the requirements with respect to the order in which they must be executed. Each task in the set must be executed once and they can by completed in any order that accords with the partial order. However, as an additional requirement, no two tasks can be executed at the same time (i.e. no two tasks can be active for the same process instance at the same time).*

Implementation: CPMF allows to define process models based on the dependencies of the activities. In two sequential activities, the second one depends on the first one. Defining a partial order between the activities is not possible in CPMF, because principally CPMF does not based its process model on the ordering of activities. Another reason for not supporting this pattern is the incapability of the supporting tool to prevent the execution of two tasks at the same time.

Evaluated rating: *Support missing (-)*

Pattern 18 (Milestone)

Description: *A task is only enabled when the process instance (of which it is part) is in a specific state (typically a parallel branch). The state is assumed to be a specific execution point (also known as a milestone) in the process model. When this execution point is reached the nominated task can be enabled. If the process instance has progressed beyond this state, then the task cannot be enabled now or at any future time (i.e. the deadline has expired). Note that the execution does not influence the state itself, i.e. unlike normal control-flow dependencies it is a test rather than a trigger.*

Implementation: CPMF allows to develop a separate lifecycle for each activity. This way the lifecycle for each parent and child activity is defined, where events are propagated between them for the transition of the states. A particular state in any activity can only be achieved, based on the events and its current current state. This allows to enable implement this pattern.

Evaluated rating: *Direct support (+)*

Pattern 19 (Cancel Task)

Description: *An enabled task is withdrawn prior to it commencing execution. If the task has started, it is disabled and, where possible, the currently running instance is halted and removed.*

Implementation: Each activity in the CPMF process model can have a separate lifecycle. If this lifecycle defines the possibility to cancel the execution of an activity, it is achievable. The default lifecycle of each activity in CPMF allows to terminate an activity after enablement. This means that it can be terminated even prior to its start of execution.

Evaluated rating: *Direct support (+)*

Pattern 20 (Cancel Case)

Description: *A complete process instance is removed. This includes currently executing tasks, those which may execute at some future time and all sub-processes. The process instance is recorded as having completed unsuccessfully.*

Implementation: Each process in the CPMF process model is contained in an activity. The lifecycle of the containing activity defines the overall lifecycle of the process. Each activity in the CPMF process model can have a separate lifecycle. If this lifecycle defines the possibility to cancel the execution of an activity, it is achievable. The default lifecycle of each activity in CPMF allows to terminate an activity after enablement. This means that it can be terminated even prior to its start of execution. Termination of an activity means the termination of the contained process.

Evaluated rating: *Direct support (+)*

Pattern 21 (Structured Loop)

Description: *The ability to execute a task or sub-process repeatedly. The loop has either a pre-test or post-test condition associated with it that is either evaluated at the beginning or end of the loop to determine whether it should continue. The looping structure has a single entry and exit point.*

Implementation: Each activity in CPMF allows for multiple iterations. The number of iterations for an activity can be specified at design time, but they can also be updated during the execution of the activity. Each activity can specify the pre-conditions and post-conditions. These conditions are evaluated for each iteration. An activity can specify indefinite iterations by giving a special value (-1) to its iteration property. In this case, a condition can be used for specifying the termination condition for an activity's execution.

Evaluated rating: *Direct support (+)*

Pattern 22 (Recursion)

Description: *The ability of a task to invoke itself during its execution or an ancestor in terms of the overall decomposition structure with which it is associated.*

Implementation: Even though an activity in CPMF can be executed for multiple iterations, it does not provide sufficient support for recursion. The core process modeling approach does not interfere with this behavior. It is not implemented in the prototype that accompanies the CPMF framework. For such a support, the prototype has to offer the possibility to store the state for each iteration of an activity during its execution.

Evaluated rating: *Support missing (-)*

Pattern 23 (Transient Trigger)

Description: *The ability for a task instance to be triggered by a signal from another part of the process or from the external environment. These triggers are transient in nature and are lost if not acted on immediately by the receiving task. A trigger can only be utilized if there is a task instance waiting for it at the time it is received.*

Implementation: Triggering of each activity in CPMF is based on the event management system. Each activity defines event listeners in the required contracts so as to listen to the events. The required contract of an activity differentiate between the transient and the persistent triggers. Transient triggers are not logged by the activity and can be acted upon immediately, if the pre-conditions are met. If the pre-conditions are met later on, this event is considered lost, as it is not logged.

Evaluated rating: *Direct support (+)*

Pattern 24 (Persistent Trigger)

Description: *The ability for a task to be triggered by a signal from another part of the process or from the external environment. These triggers are persistent in form and are retained by the process until they can be acted on by the receiving task.*

Implementation: Triggering of each activity in CPMF is based on the event management system. Each activity defines event listeners in the required contracts so as to listen to the events. The required contract of an activity differentiate between the transient and the persistent triggers. Persistent triggers are logged by the activity and can be acted upon immediately and even after some time, when the pre-conditions are met. If the pre-conditions are met later on, this event is considered received by the activity, as it is logged.

Evaluated rating: *Direct support (+)*

Pattern 25 (Cancel Region)

Description: *The ability to disable a set of tasks in a process instance. If any of the tasks are already executing (or are currently enabled), then they are withdrawn. The tasks need not be a connected subset of the overall process model.*

Implementation: Each activity in the CPMF process model can have a separate lifecycle. If this lifecycle defines the possibility to cancel the execution of an activity, it is achievable. The default lifecycle of each activity in CPMF allows to terminate an activity after enablement. In order to terminate a set of activities, it must be done individually, one by one. It can also be achieved by grouping this set of activities under one process, this allows to terminated all these activities through the termination of the containing process.

Evaluated rating: *Partial support (+/-)*

Pattern 26 (Cancel Multiple Instance Task)

Description: *Within a given process instance, multiple instances of a task can be created. The required number of instances is known at design time. These instances are independent of each other and run concurrently. At any time, the multiple instance task can be canceled and any instances which have not completed are withdrawn. Task instances that have already completed are unaffected.*

Implementation: CPMF allows multiple instances of an activity in its process models. Each of these instances have their own properties. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Each activity instance follows its own lifecycle, separate from other activity instances. This allows concurrent execution of each activity instance. If this lifecycle defines the possibility to cancel the execution of an activity, it is achievable. The default lifecycle of each activity in CPMF allows to terminate an activity after enablement. Terminating an activity that is already enabled in a process model results in the termination of all its instance that is executing. The instances that are already complete are not affected. In order to terminate all the instances, they all need to be terminated manually one by one.

Evaluated rating: *Partial support (+/-)*

Pattern 27 (Complete Multiple Instance Task)

Description: *Within a given process instance, multiple instances of a task can be created. The required number of instances is known at design time. These instances are independent of each other and run concurrently. It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered. During the course of execution, it is possible that the task needs to be forcibly completed such that any remaining instances are withdrawn and the thread of control is passed to subsequent tasks.*

Implementation: CPMF allows multiple instances of an activity in its process models. Each of these instances have their own properties. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Each activity instance follows its own lifecycle, separate from other activity instances. This allows concurrent execution of each activity instance. If this lifecycle defines the possibility to specify the 'complete' status of the execution of an activity, it is achievable. The default lifecycle of each activity in CPMF allows to specify the complete state an activity. Completion of an activity that is already enabled in a process model results in the completion of all its instance that is executing. The instances that are already complete are not affected. In order to specify the completion of all the instances, they all need to be specified manually one by one.

Evaluated rating: *Partial support (+/-)*

Pattern 28 (Blocking Discriminator)

Description: *The convergence of two or more branches into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when the first active incoming branch has been enabled. The Blocking Discriminator construct resets when all active incoming branches have been enabled once for the same process instance. Subsequent enablements of incoming branches are blocked until the Blocking Discriminator has reset.*

Implementation: In CPMF, such a control flow structure can be achieved when a single activity is bound to two or more precedent activities. In this case, the required contract of the subsequent activity specifies the behavior (i.e. OR behavior) through a pre-condition. Events triggered by each of the precedent activities bound to the single subsequent activity are listened. If an event of enablement is received the subsequent activity can start its execution. Before the start of the execution, the event is logged. This blocks any further input, until it is reset. It can be reset when all the incoming branches have been enabled once.

Evaluated rating: *Direct support (+)*

Pattern 29 (Canceling Discriminator)

Description: *The convergence of two or more branches into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when the first active incoming branch has been enabled. Triggering the Cancelling Discriminator also cancels the execution of all of the other incoming branches and resets the construct.*

Implementation: Such a control flow structure can be achieved when a single activity is bound to two or more precedent activities in CPMF. In this case, the required contract of the subsequent activity specifies the behavior (i.e. OR behavior) through a pre-condition. Events triggered by each of the precedent activities bound to the single subsequent activity are listened. If an event of enablement is received the subsequent activity can start its execution. Before the start of the execution, the event is logged. Events from other branches are canceled if already one of the branch event is logged. The contract is reset when events are received from all branches.

Evaluated rating: *Direct support (+)*

Pattern 30 (Structured Partial Join)

Description: *The convergence of two or more branches (say m) into a single subsequent branch following a corresponding divergence earlier in the process model such that the thread of control is passed to the subsequent branch when n of the incoming branches have been enabled where n is less than m. Subsequent enablements of incoming branches do not result in the thread of control being passed on. The join construct* *resets when all active incoming branches have been enabled. The join occurs in a structured context, i.e. there must be a single Parallel Split construct earlier in the process model with which the join is associated and it must merge all of the branches emanating from the Parallel Split. These branches must either flow from the Parallel Split to the join without any splits or joins or be structured in form (i.e. balanced splits and joins).*

Implementation: The preconditions in the required contract of the subsequent activity can be used for defining a partial or complete join. If a partial join is specified, the control is transfered for the defined partial input contracts. The rest of the incoming inputs would become inconsequential. In order to reset the contract, when all the active incoming inputs are received, the log is cleared, which finally is used to reset the contract. In order to support structured aspect of this join, the structure of parallel split earlier in the model that is associated with the this parallel join has to be managed manually. No direct support is present to take care of this association.

Evaluated rating: *Partial support (+/-)*

Pattern 31 (Blocking Partial Join)

Description: *The convergence of two or more branches (say m) into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when n of the incoming branches has been enabled (where $2 = n < m$). The join construct resets when all active incoming branches have been enabled once for the same process instance. Subsequent enablements of incoming branches are blocked until the join has reset.*

Implementation: The preconditions in the required contract of the subsequent activity can be used for defining a partial or complete join. If a partial join is specified, the control is transfered for the defined partial input contracts. The rest of the incoming inputs would be blocked by this pattern. In order to reset the contract, when all the incoming inputs are received, the log is cleared, which finally is used to reset the contract.

Evaluated rating: *Partial support (+/-)*

Pattern 32 (Canceling Partial Join)

Description: *The convergence of two or more branches (say m) into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when n of the incoming branches have been enabled where n is less than m. Triggering the join also cancels the execution of all of the other incoming branches and resets the construct.*

Implementation: The preconditions in the required contract of the subsequent activity can be used for defining a partial or complete join. If a partial join is specified, the control is transfered for the defined partial input contracts. The rest of the incoming inputs would be canceled by this pattern. In order to reset the contract, when all the incoming inputs are received, the log is cleared, which finally is used to reset the contract.

Evaluated rating: *Partial support (+/-)*

Pattern 33 (Generalized AND-Join)

Description: *The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled. Additional triggers received on one or more branches between firings of the join persist and are retained for future firings. Over time, each of the incoming branches should deliver the same number of triggers to the AND-join construct (although obviously, the timing of these triggers may vary).*

Implementation: The preconditions in the required contract of the subsequent activity can be used for defining a join. The control is transfered from the input contracts, once events are listened from all the branches. Each event is logged. Once events are received from all the branches, the activity can be triggered and the log is cleared for one cycle. Additional events received from any of the branches are retained in the log for future enablements.

Evaluated rating: *Partial support (+/-)*

Pattern 34 (Static Partial Join for Multiple Instances)

Description: *Within a given process instance, multiple concurrent instances of a task (say m) can be created. The required number of instances is known when the first task instance commences. Once n of the task instances have completed (where n is less than m), the next task in the process is triggered. Subsequent completions of the remaining m-n instances are inconsequential, however all instances must have completed in order for the join construct to reset and be subsequently re-enabled.*

Implementation: Multiple instances of an activity in the process model can be created. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Each of these instances have their own properties and the same contracts as defined in the activity definition. Each activity instance follows its own lifecycle, separate from other activity instances, thus allowing concurrent execution of each activity instance. The preconditions in the required contract of the subsequent activity can be used for defining a partial or complete join. If a partial join is specified, the control is transfered for the defined partial input contracts. The rest of the incoming inputs would become inconsequential. In order to reset the contract, when all the incoming inputs are received, the log is cleared, which finally is used to reset the contract.

Evaluated rating: *Direct support (+)*

Pattern 35 (Canceling Partial Join for Multiple Instances)

Description: *Within a given process instance, multiple concurrent instances of a task (say m) can be created. The required number of instances is known when the first task instance commences. Once n of the task instances have completed (where n is less than m), the next task in the process is triggered and the remaining m-n instances are canceled.*

Implementation: Multiple instances of an activity in the process model can be created. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Each of these instances have their own properties and the same contracts as defined in the activity definition. Each activity instance follows its own lifecycle, separate from other activity instances, thus allowing concurrent execution of each activity instance. The preconditions in the required contract of the subsequent activity can be used for defining a partial or complete join. If a partial join is specified, the control is transfered for the defined partial input contracts. The rest of the incoming inputs however can not be canceled by the subsequent activity. This does not mean that it is not achievable. The amount of effort required for this would be relatively high. A special lifecycle needs to be defined for the precedent activity implementations that allows the subsequent activity implementation to trigger a cancel event for it. Thus it can be achieved, but is not directly supported by the prototype.

Evaluated rating: *Support missing (-)*

Pattern 36 (Dynamic Partial Join for Multiple Instances)

Description: *Within a given process instance, multiple concurrent instances of a task can be created. The required number of instances may depend on a number of runtime factors, including state data, resource availability and inter-process communications and is not known until the final instance has completed. At any time, whilst instances are running, it is possible for additional instances to be initiated providing the ability to do so had not been disabled. A completion condition is specified which is evaluated each time an instance of the task completes. Once the completion condition evaluates to true, the next task in the process is triggered. Subsequent completions of the remaining task instances are inconsequential and no new instances can be created.*

Implementation: Multiple instances of an activity in the process model can be created. Activity implementation in the instantiation phase process model specify the allowed number of instances for each implementation. Each of these instances have their own properties and the same contracts as defined in the activity definition. Each activity instance follows its own lifecycle, separate from other activity instances, thus allowing concurrent execution of each activity instance. The preconditions in the required contract of the subsequent activity can be used for defining a partial or complete join. If a partial join is specified, the control is transfered for the defined partial input contracts. As the precondition in the required contract of the subsequent activity defines the partial join, it can also include a dynamic condition that needs to be evaluated before the control is transfered. The rest of the incoming inputs would become inconsequential. In order to reset the contract, when all the incoming inputs are received, the log is cleared, which finally is used to reset the contract.

Evaluated rating: *Direct support (+)*

Pattern 37 (Local Synchronizing Merge)

Description: *The convergence of two or more branches which diverged earlier in the process into a single subsequent branch such that the thread of control is passed to the subsequent branch when each active incoming branch has been enabled. Determination of how many branches require synchronization is made on the basis on information locally available to the merge construct. This may be communicated directly to the merge by the preceding diverging construct or alternatively it can be determined on the basis of local data such as the threads of control arriving at the merge.*

Implementation: The preconditions in the required contract of the subsequent activity can be used for defining such a merge pattern. The control is transfered from the input contracts, once events are listened from all the synchronization branches. The number of synchronization branches is communicated to this required contract through the event broker of the parent activity. The event broker of the parent activity keeps a log of the associated diverging construct present in the process model.

Evaluated rating: *Direct support (+)*

Pattern 38 (General Synchronizing Merge)

Description: *The convergence of two or more branches which diverged earlier in the process into a single subsequent branch such that the thread of control is passed to the subsequent branch when either (1) each active incoming branch has been enabled or (2) it is not possible that any branch that has not yet been enabled will be enabled at any future time.*

Implementation: Current implementation of CPMF does not allow the possibility to an activity to know in advance, whether a branch to one of its input contracts will never be enabled at any future time.

Evaluated rating: *Support missing (-)*

Pattern 39 (Critical Section)

Description: *Two or more connected sub-graphs of a process model are identified as "critical sections". At runtime for a given process instance, only tasks in one of these "critical sections" can be active at any given time. Once execution of the tasks in one "critical section" commences, it must complete before another "critical section" can commence.*

Implementation: Their is not support in the CPMF implementation to pause one of the execution branches by another execution branch of a process model. This does not mean that such a pattern can not be implemented. It would require to develop the process model in a way, where the two execution branches start with an OR-split type pattern. In this way, only one of the execution branches can be active at a given time. A reset mechanism for the pattern can be achieved through a corresponding OR-join pattern, which resets the original OR-split for further processing.

Evaluated rating: *Support missing (-)*

Pattern 40 (Interleaved Routing)

Description: *Each member of a set of tasks must be executed once. They can be executed in any order but no two tasks can be executed at the same time (i.e. no two tasks can be active for the same process instance at the same time). Once all of the tasks have completed, the next task in the process can be initiated.*

Implementation: CPMF process model can execute a process model sequentially (restricting two activities to execute at the same time) in a way where a dependency is introduced between all the activities of the process model. This will take away the possibility to have a dynamic ordering of activities without any dependency between them. It is not possible in the current implementation of CPMF to block the parallel execution of two activities from the same process model, if there is no dependency between them.

Evaluated rating: *Support missing (-)*

Pattern 41 (Thread Merge)

Description: *At a given point in a process, a nominated number of execution threads in a single branch of the same process instance should be merged together into a single thread of execution.*

Implementation: A process instance in CPMF is actually an activity instance that contains the process. Hence a process thread is an activity in execution. Thread merge is handled by CPMF similar to the activity merge.

Evaluated rating: *Direct support (+)*

Pattern 42 (Thread Split)

Description: *At a given point in a process, a nominated number of execution threads can be initiated in a single branch of the same process instance.*

Implementation: A process instance in CPMF is actually an activity instance that contains the process. Hence a process thread is an activity in execution. Thread split is handled by CPMF similar to the activity split.

Evaluated rating: *Direct support (+)*

Pattern 43 (Explicit Termination)

Description: *A given process (or sub-process) instance should terminate when it reaches a nominated state. Typically this is denoted by a specific end node. When this end node is reached, any remaining work in the process instance is canceled and the overall process instance is recorded as having completed successfully, regardless of whether there are any tasks in progress or remaining to be executed.*

Implementation: CPMF process model allows to define a customizable life-cycle for each activity. In case a user does not define a life-cycle, a default life-cycle is followed by the activity. This life-cycle allows to terminate the activity from one or more states. Propagation of event from child activities to their parent activities and parent activities to their child activities ensures the management of respective lifecycles. When a parent activity is complete or terminated, this event is propagated to all its child activities. All child activities that are not complete or terminated yet, listen to the event and change their state according to the respective event. This ensures explicit termination of an activity within the process model.

Evaluated rating: *Direct support (+)*

1.3 Workflow Resource Patterns

Pattern 1 (Direct Distribution)

Description: *The ability to specify at design time the identity of the resource(s) to which instances of this task will be distributed at runtime.*

Implementation: The final design of the process model is complete in the instance process model, where each activity is associated with one or more actors, who perform them.

Evaluated rating: *Direct support (+)*

Pattern 2 (Role-Based Distribution)

Description: *The ability to specify at design-time one or more roles to which instances of this task will be distributed at runtime. Roles serve as a means of grouping resources with similar characteristics. Where an instance of a task is distributed in this way, it is distributed to all resources that are members of the role(s) associated with the task.*

Implementation: It is possible to know the roles associated with each actor. This allows to allocate the tasks to the actors based on the roles that they are performing.

Evaluated rating: *Direct support (+)*

Pattern 3 (Deferred Distribution)

Description: *The ability to specify at design-time that the identification of the resource(s) to which instances of this task will be distributed will be deferred until runtime.*

Implementation: It is possible to execute partial process models in CPMF. This allows to add the details during the execution. These details are added through the project management dashboard. However, the exact activity can not be performed unless a resource is allocated to it. Thus the resource needs to be allocated to the activity before its execution.

Evaluated rating: *Direct support (+)*

Pattern 4 (Authorization)

Description: *The ability to specify the range of privileges that a resource possesses in regard to the execution of a process. In the main, these privileges define the range of actions that a resource can initiate when undertaking work items associated with tasks in a process.*

Implementation: The use of 'responsiblities' in CPMF process model, allows to define the range of privileges associated with each role. These roles are then played by actors. Thus each actor plays a role with well specified privileges.

Evaluated rating: *Direct support (+)*

Pattern 5 (Separation of Duties)

Description: *The ability to specify that two tasks must be executed by different resources in a given case.*

Implementation: Two tasks can be associated to two different actors in CPMF. Association of a task to an actor is deferred till instantiation phase. In this phase each activity is independent of other activities.

Evaluated rating: *Direct support (+)*

Pattern 6 (Case Handling)

Description: *The ability to allocate the work items within a given case to the same resource at the time that the case is commenced.*

Implementation: A case is a sub-process in CPMF framework that is contained within an activity. Thus the composite activity containing this sub-process can be allocated to a resource during the execution of the process model, in the same way as other activities.

Evaluated rating: *Direct support (+)*

Pattern 7 (Retain Familiar)

Description: *Where several resources are available to undertake a work item, the ability to allocate a work item within a given case to the same resource that undertook a preceding work item.*

Implementation: Actors associated to any activity can be accessed by the framework. This allows to access the actor associated with the previous activity and associate it with the current activity.

Evaluated rating: *Direct support (+)*

Pattern 8 (Capability-Based Distribution)

Description: *The ability to distribute work items to resources based on specific capabilities that they possess. Capabilities (and their associated values) are recorded for individual resources as part of the organizational model.*

Implementation: Capabilities are associated with each actor. This allows to record the skill set of each individual at disposal for the execution of the process. It is possible to allocate an actor to an activity, based on his/her capabilities.

Evaluated rating: *Direct support (+)*

Pattern 9 (History-Based Distribution)

Description: *The ability to distribute work items to resources on the basis of their previous execution history.*

Implementation: CPMF interpreter keeps the execution history for each actor. But this is valid only for the execution of the current process model. There is no database attached to the interpreter to keep track of all his execution history prior to the current process model. Thus the activities can be associated with an actor based on his/her previous execution history (within the current process model).

Evaluated rating: *Partial support (+/-)*

Pattern 10 (Organizational Distribution)

Description: *The ability to distribute work items to resources based their position within the organisation and their relationship with other resources.*

Implementation: CPMF interpreter does not keep track of the organizational break down structure. Thus it is not possible to relate two actors in terms of their organizational hierarchy.

Evaluated rating: *Support missing (-)*

Pattern 11 (Automatic Execution)

Description: *The ability for an instance of a task to execute without needing to utilise the services of a resource.*

Implementation: CPMF framework allows for three different types of activities: manual, semi-automatic and automatic. Automatic activities can be executed by the interpreter without the need of any human intervention.

Evaluated rating: *Direct support (+)*

Pattern 12 (Distribution by Offer - Single Resource)

Description: *The ability to distribute a work item to a selected individual resource on a non-binding basis.*

Implementation: Current implementation of the CPMF framework binds each actor to perform the allocated activities. It is considered to be binding and he is not given an opportunity to reject the allocated activities. However project manager has the option to re-allocate an activity to another actor.

Evaluated rating: *Support missing (-)*

Pattern 13 (Distribution by Offer - Multiple Resources)

Description: *The ability to distribute a work item to a group of selected resources on a non-binding basis.*

Implementation: Current implementation of the CPMF framework binds each actor to perform the allocated activities. It is considered to be binding and he is not given an opportunity to reject the allocated activities. However project manager has the option to re-allocate an activity to another actor. This is the same case for single or multiple actors.

Evaluated rating: *Support missing (-)*

Pattern 14 (Distribution by Allocation - Single Resource)

Description: *The ability to distribute a work item to a specific resource for execution on a binding basis.*

Implementation: All activities allocated to an actor in the CPMF process framework are considered binding.

Evaluated rating: *Direct support (+)*

Pattern 15 (Random Allocation)

Description: *The ability to allocate work items to a selected resource chosen from a group of eligible resources on a random basis.*

Implementation: It is possible to allocate the activities to any random actor that has the required capabilities to perform the role.

Evaluated rating: *Direct support (+)*

Pattern 16 (Round Robin Allocation)

Description: *The ability to allocate a work item to a selected resource chosen from a group of eligible resources on a cyclic basis.*

Implementation: It is possible to allocate the activities to any random actor that has the required capabilities to perform the role.

Evaluated rating: *Direct support (+)*

Pattern 17 (Shortest Queue)

Description: *The ability to allocate a work item to a selected resource chosen from a group of eligible resources on the basis of having the shortest work queue.*

Implementation: Every actor maintains a list of allocated activities to it. However the approach does not directly provide a mechanism to compare the working queues of different actors. With a little programmatic extension, it is possible to compare the working queues of different actors and allocate a certain activity to the actor having the shortest queue.

Evaluated rating: *Partial support (+/-)*

Pattern 18 (Early Distribution)

Description: *The ability to advertise and potentially distribute a work items to resources ahead of the moment at which it is actually enabled.*

Implementation: Activities can be allocated to actors as early as in the instance process model. That means, even before the execution of the process model.

Evaluated rating: *Direct support (+)*

Pattern 19 (Distribution on Enablement)

Description: *The ability to advertise and distribute a work items to resources at the moment that the task to which it corresponds is enabled for execution.*

Implementation: It is possible to allocate the activities to an actor during the execution of the process model. This is handled through the project management dashboard. However the exact activity can not be executed unless it has been allocated to an actor.

Evaluated rating: *Direct support (+)*

Pattern 20 (Late Distribution)

Description: *The ability to advertise and distribute work items to resources after the task to which the work item corresponds has been enabled for execution.*

Implementation: It is possible to allocate the activities to an actor during the execution of the process model. This is handled through the project management dashboard. However the exact activity can not be executed unless it has been allocated to an actor. Thus this case can not arrive in CPMF, where the activity shall wait for its allocation for execution.

Evaluated rating: *Support missing (-)*

Pattern 21 (Resource Initiated Allocation)

Description: *The ability for a resource to commit to undertake a work item without needing to commence working on it immediately.*

Implementation: An allocation of an activity to an actor can be done as early as in the instance process model, that means even before the execution of the process model. This allocation can be initiated by the actor itself (if he has the privileges to assign activities).

Evaluated rating: *Direct support (+)*

Pattern 22 (Resource-Initiated Execution - Allocated Work Item)

Description: *The ability for a resource to commence work on a work item that is allocated to it.*

Implementation: An actor can access his own specific area on the project management dashboard. This allows him to manage all the activities allocated to him/her. From this portal, he is able to commence the execution of the activities allocated to him/her.

Evaluated rating: *Direct support (+)*

Pattern 23 (Resource-Initiated Execution - Offered Work Item)

Description: *The ability for a resource to select a work item offered to it and commence work on it immediately.*

Implementation: An actor can access his own specific area on the project management dashboard. This allows him to manage all the activities allocated to him/her. From this portal, he is able to commence the execution of the activities allocated to him/her.

Evaluated rating: *Direct support (+)*

Pattern 24 (System-Determined Work Queue Content)

Description: *The ability of the system to order the content and sequence in which work items are presented to a resource for execution.*

Implementation: Activities allocated to an actor may have dependencies between them. In this case they are sequenced by the system and must be performed in the right sequence.

Evaluated rating: *Direct support (+)*

Pattern 25 (Resource-Determined Work Queue Content)

Description: *The ability for resources to specify the format and content of work items listed in the work queue for execution.*

Implementation: In case where the activities allocated to an actor do not have any dependencies between them, an actor is at liberty to arrange the sequence of the activities allocated to him.

Evaluated rating: *Direct support (+)*

Pattern 26 (Selection Autonomy)

Description: *The ability for resources to select a work item for execution based on its characteristics and their own preferences.*

Implementation: In case where the activities allocated to an actor do not have any dependencies between them, an actor is at liberty to arrange the sequence of the activities allocated to him. This sequence of activities can be based on the personal preferences.

Evaluated rating: *Direct support (+)*

Pattern 27 (Delegation)

Description: *The ability for a resource to allocate an un-started work item previously allocated to it (but not yet commenced) to another resource.*

Implementation: The actor (having the privileges for activity allocation) can allocate and re-allocate the activities to any other actor. Any actor that does not have the necessary privileges has to contact the process owner.

Evaluated rating: *Direct support (+)*

Pattern 28 (Escalation)

Description: *The ability of a system to distribute a work item to a resource or group of resources other than those it has previously been distributed to in an attempt to expedite the completion of the work item.*

Implementation: Activities can be re-allocated to any other actor or team, based on any intention. For a complex re-allocation of activities, activity adaptations in CPMF process model allows to reconfigure the process models completely.

Evaluated rating: *Direct support (+)*

Pattern 29 (De-allocation)

Description: *The ability of a resource (or group of resources) to relinquish a work item which is allocated to it (but not yet commenced) and make it available for distribution to another resource or group of resources.*

Implementation: All actors having the privileges to allocate the activities can also de-allocate the activities from a certain actor or team. However, for an actor/team that does not possess the privileges has to contact the process owner.

Evaluated rating: *Direct support (+)*

Pattern 30 (Stateful Reallocation)

Description: *The ability of a resource to allocate a work item that they are currently executing to another resource without loss of state data.*

Implementation: Activity adaptations in CPMF framework takes care of the transfer of state between the two activities. Activity adaptations is handled through replacing one activity (or sub-activity) with another. In this case, the state is transfered through two mechanisms: 1) hard-coded links between the properties of the two activities and 2) interactive linking where the properties of the two activities are presented to the process owner in project management dashboard. He can link the properties where state needs to be transfered.

Evaluated rating: *Direct support (+)*

Pattern 31 (Stateless Reallocation)

Description: *The ability for a resource to reallocate a work item that it is currently executing to another resource without retention of state.*

Implementation: Activity adaptations in CPMF framework takes care of the transfer of state between the two activities. Activity adaptations is handled through replacing one activity (or sub-activity) with another. In this case, the state is transfered through two mechanisms: 1) hard-coded links between the properties of the two activities and 2) interactive linking where the properties of the two activities are presented to the process owner in project management dashboard. He can link the properties where state needs to be transfered. Process owner can choose interactive mechanisms and not link any of the properties for a stateless adaptation.

Evaluated rating: *Direct support (+)*

Pattern 32 (Suspension/Resumption)

Description: *The ability for a resource to suspend and resume execution of a work item.*

Implementation: Each actor can access his specific working area on the project management dashboard. He can the ability to manage all the activities allocated to him. This project management dashboard allows the actor to carry out the transitions to an activity, based on the lifecycle of the activity. The default lifecycle of every activity allows to pause/resume an activity. Thus an actor can pause and resume all activities allocated to him.

Evaluated rating: *Direct support (+)*

Pattern 33 (Skip)

Description: *The ability for a resource to skip a work item allocated to it and mark the work item as complete.*

Implementation: Each actor can access his specific working area on the project management dashboard. He can the ability to manage all the activities allocated to him. This project management dashboard allows the actor to carry out the transitions to an activity, based on the lifecycle of the activity. The default lifecycle of every activity allows to the 'complete' state of an activity. Thus an actor can change the state of any or all the activities allocated to him to complete.

Evaluated rating: *Direct support (+)*

Pattern 34 (Redo)

Description: *The ability for a resource to redo a work item that has previously been completed in a case. Any subsequent work items (i.e. work items that correspond to subsequent tasks in the process) must also be repeated.*

Implementation: Redoing an activity is possible in CPMF framework only if the lifecycle of the activity permits it. For a process model where certain activities need to be repeated, the lifecycle of the activities should be developed in the appropriate manner. This also applies for the lifecycle of the subsequent activities. The default lifecycle of activities does not support it, but it can be performed by customizing the lifecycle, which is allowed by the framework.

Evaluated rating: *Partial support (+/-)*

Pattern 35 (Pre-Do)

Description: *The ability for a resource to execute a work item ahead of the time that it has been offered or allocated to resources working on a given case. Only work items that do not depend on data elements from preceding work items can be "pre-done".*

Implementation: Any activity allocated to an actor can be performed by the actor if its dependencies are already met. If the dependencies are met, the state of the activity turns from waiting to ready state. When an activity is in ready state, the associated actor can perform it even before the scheduled time.

Evaluated rating: *Direct support (+)*

Pattern 36 (Commencement on Creation)

Description: *The ability for a resource to commence execution on a work item as soon as it is created.*

Implementation: Any activity allocated to an actor can be performed by the actor if its dependencies are already met. If the dependencies are met, the state of the activity turns from waiting to ready state. Only, when an activity is in ready state, the associated actor can perform it. Thus it can not be performed unless an actor is associated or unless the dependencies are met.

Evaluated rating: *Support missing (-)*

Pattern 37 (Commencement on Allocation)

Description: *The ability to commence execution on a work item as soon as it is allocated to a resource.*

Implementation: Any activity allocated to an actor can be performed by the actor if its dependencies are already met. If the dependencies are met, the state of the activity turns from waiting to ready state. Only, when an activity is in ready state, the associated actor can perform it. Thus it can not be performed unless an actor is associated or unless the dependencies are met.

Evaluated rating: *Direct support (+)*

Pattern 38 (Piled Execution)

Description: *The ability to initiated the next instance of a task (perhaps in a different case) once the previous one has completed with all associated work items* *being allocated to the same resource. The transition to Piled Execution mode is at the instigation of an individual resource. Only one resource can be in Piled Execution mode for a given task at any time.*

Implementation: Once an activity is complete, it outputs the artifacts provided by it. The next activity in sequence if dependent on it, can start its execution as soon as all its dependencies are met. The trigger of the next activity can be set to the completion of all dependencies, human intervention or both.

Evaluated rating: *Partial support (+/-)*

Pattern 39 (Chained Execution)

Description: *The ability to automatically start the next work item in a case once the previous one has completed. The transition to Chained Execution mode is at the instigation of the resource.*

Implementation: Once an activity is complete, it outputs the artifacts provided by it. The next activity in sequence if dependent on it, can start its execution as soon as all its dependencies are met. The trigger of the next activity can be set to the completion of all dependencies, human intervention or both.

Evaluated rating: *Partial support (+/-)*

Pattern 40 (Configurable Unallocated Work Item Visibility)

Description: *The ability to configure the visibility of unallocated work items by process participants.*

Implementation: Process participants have access to all the activities allocated to them through the project management dashboard. Only process owner/administrator has access to all the activities in the process. Thus for actors, it is not possible to configure or even view the state of the activities not allocated to them.

Evaluated rating: *Support missing (-)*

Pattern 41 (Configurable Allocated Work Item Visibility)

Description: *The ability to configure the visibility of allocated work items by process participants.*

Implementation: Process participants have access to all the activities allocated to them through the project management dashboard. Only process owner/administrator has access to all the activities in the process. All actors that have the required privileges can configure the visibility of the activities allocated to them.

Evaluated rating: *Direct support (+)*

Pattern 42 (Simultaneous Execution)

Description: *The ability for a resource to execute more than one work item simultaneously.*

Implementation: Process participants have access to all the activities allocated to them through the project management dashboard. Thus the actor has access to the states and transitions of all the allocated activities. He/she has the possibility to execute multiple concurrent activities in parallel, if their dependencies are already met.

Evaluated rating: *Direct support (+)*

Pattern 43 (Additional Resources)

Description: *The ability for a given resource to request additional resources to assist in the execution of a work item that it is currently undertaking.*

Implementation: CPMF framework allows to adapt the activities during their execution. The framework also allows a communication service between the process participants. An actor that requires addition resources for assistance has to ask the process owner/administrator for allocating more actors to the current activity. Process owner can adapt the process on the fly, so as to allocate multiple actors to it. Once the process is adapted, he/she can allocate additional actors or teams.

Evaluated rating: *Direct support (+)*

Appendix B

Process Model Constructs

Constructs	SPEM	xSPEM	MODAL	BPMN	BPEL	EPCs	YAWL	Little-JIL
Process	+	+	+	+	+	+	+	+
Activity Type	-	-	-	-	-	-	-	-
Composite Activity	+	+	+	+	+	+	+	+
Primitive Activity	+	+	+	+	+	+	+	+
Responsibility	-	-	-	-	-	-	-	-
Role	+	+	+	+	-	+	+	+
Team	+	+	+	+	-	+	+	+
Actor	-	-	-	+	-	+	+	+
Tool	+	+	+	+	-	+	+	+
Goal	-	-	+	-	-	-	-	-
Guideline	+	+	+	-	-	-	-	-
Data-flow	-	-	-	+	+	+	+	+
Control-flow	+	+	+	+	+	+	+	+
Artifact Specification	-	-	-	-	-	-	-	-
Artifact Metamodel	-	-	-	-	-	-	-	-
Artifact	+	+	+	+	+	+	+	+
Event	-	-	-	+	+	+	+	+
Message Event	-	-	-	+	+	-	+	+
State	-	-	+	+	+	-	+	+
Conditions	+	+	+	+	+	-	+	+

Table B.1 – Corresponding constructs support in state of the art