La Découverte de WorkflowTransactionnel pour la Fiabilisation desExécutions

Walid Gaaloul

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

La Découverte de Workflow Transactionnel pour la Fiabilisation des Exécutions

THÈSE

présentée et soutenue publiquement le 03/11/2006

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Walid Gaaloul

Composition du jury

Rapporteurs :	M. Boualem Benatallah, Professeur à l'Université de New South Wales, Sydney, Australia M. Farouk Toumani , Professeur à l'Université Blaise Pascal, Clermont Ferrand
Examinateurs :	M. Amedeo Napoli, Directeur de recherche CNRS, Nancy M. Claude Godart, Professeur à l'Université Henri Poincaré, Nancy 1 M. Karim Baïna, Professeur-Assistant, ENSIAS, Rabat, Maroc

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

Une évolution continue des paramètres, des contraintes et des besoins du procédé métier, non complètement prévisible initialement, exige des systèmes de gestion de procédés une conception continue et un modèle de procédé fiable. Dans cette thèse, nous nous intéressons à assurer une conception réactive par l'analyse des traces d'exécutions assurant une re-ingénierie du procédé métier et une fiabilisation des exécutions.

Pour ce faire, nous introduisons d'abord un modèle de workflow transactionnel qui étend les systèmes de workflows en les fusionnant avec les modèles transactionnels avancés. Ce modèle, formellement décrit, permet de mieux spécifier les mécanismes de recouvrement nécessaires à une fiabilisation des exécutions. Nous proposons, par la suite, des techniques d'analyse de traces d'exécutions pour la découverte de workflow et l'amélioration de leurs comportements transactionnels. Notre approche commence par la collecte des traces d'exécution. Nous construisons, ensuite, par des techniques d'analyse statistique, une représentation intermédiaire spécifiant des dépendances élémentaires entre les activités. Ces dépendances sont raffinées pour découvrir le modèle de workflow transactionnel. L'analyse des disparités entre le modèle découvert et le modèle initialement conçu nous permet de détecter des lacunes (anomalies) de conception, concernant particulièrement les mécanismes de recouvrement. En fonction de ces observations, nous appliquons finalement un ensemble de règles d'amélioration et/ou de correction du schéma initial du workflow transactionnel.

La contribution majeure de notre proposition est qu'elle permet de tenir compte des besoins d'évolution du procédé observés dans la phase d'exécution. Ceci nous permet d'assurer une conception continue garantissant, parmi d'autres, des exécutions correctes et fiables.

Mots-clès : Workflow transactionnel, découverte de procédé, traces d'exécutions, re-ingénierie des procédés, fiabilité, correction.

Abstract

A continuous evolution of business process parameters, constraints and needs, hardly unforeseeable initially, requires from the business process management systems a continuous design and a reliable process model. In this thesis, we are interested in developing a reactive design through a process logs analysis ensuring process reengineering and execution reliability.

For that purpose, we introduce, first, a transactional workflow model that extends the workflow systems by merging them with the advanced transactional models. This model, which is formally described, allows to better specify recovery mechanisms necessary to ensure execution reliability. We propose, thereafter, workflow logs analysis techniques for workflow mining and transactional behavior improvement. Our approach starts by collecting workflow logs. Then, we build, by statistical analysis techniques, an intermediate representation specifying activities elementary dependencies. These dependencies are refined to mine the transactional workflow model. The analysis of the disparities between the discovered model and the initially designed model enables us to be detect design gaps (anomalies), concerning particularly the recovery mechanisms. In function of these observations, we apply a set of improvement and/or correction rules on the initially designed workflow.

The major contribution of our proposal is its ability to take into account process evolution needs observed at runtime. This allows to ensure an evolutionary workflow design guaranteeing, among others, correct and reliable execution.

Keywords : Transactional workflow, process mining, workflow logs, process reengineering, execution reliability, correction.

Remerciements

Je voudrais exprimer mes sentiments les plus spontanés envers les personnes qui sans lesquelles ce travail de thèse n'aurait pas pu voir le jour. Leur aide, accompagnement et soutien m'ont été indispensables afin de pouvoir aboutir aux contributions de ma thèse.

Je voudrais tout d'abord exprimer ma reconnaissance envers tous les membres du jury pour la grande attention qu'ils ont bien voulu porter à mon travail.

Je suis très reconnaissant à Claude Godart, mon directeur de thèse, qui m'a encadré et dirigé dans mes recherches tout au long de ces années. Je lui dis ma gratitude pour l'aide compétente qu'il m'a apportée, pour ses encouragements et pour la confiance qu'il m'a toujours témoignée.

Mes plus chaleureux remerciements vont également à Karim Baïna, qui a su m'aider pour mener à bien le présent travail. Sa disponibilité et ses conseils m'ont permis de mieux cerner les problèmes et ainsi de mieux y répondre.

Je remercie très sincèrement mes rapporteurs Boualem Benatallah et Farouk Toumani pour avoir bien accepté d'être mes rapporteurs et pour avoir bien voulu lire et évaluer mon travail de thèse. Je les remercie pour leurs lectures approfondies de mon mémoire de thèse, pour tout le temps qu'ils m'ont accordé et pour les remarques très constructives qu'ils m'ont données.

Je remercie Amedeo Napoli qui a bien voulu être mon rapporteur interne, pour son effort qu'il a bien voulu consacrer à l'évaluation de mon travail de thèse.

Je remercie l'ensemble des membres de l'equipe ECOO avec qui j'ai eu le plaisir de travailler. Je pense particulièrement à Claude Godart, Khalid Benali, François Charoy, Pascal Molli, Nacer Boudjlida, Pascal Urso, Hala Skaf-Molli, et Gérôme Canals. Mes plus amicales remerciements vont aussi aux doctorants de l'equipe, notamment mon cher frère Khaled, mon compagnon de bureau et mon meilleur conseil Sami Bhiri et celui qu'il a remplacé avec excellence Mohsen Rouached, à Gérald Oster, Adnene Guabtni, Alicia Diaz, et Ustun Yildiz qui ont permis de faire de cette expérience de thèse une expérience riche tant scientifiquement que humainement. Je tiens à remercier aussi toutes les personnes du LORIA qui travaillent dans l'ombre mais qui répondent toujours présentes quand nous avons besoin d'elles, et particulièrement Antoinette Courrier.

Merci à tous mes amis, en particulier Nizar, Abdelhalim, Zied, Mohammed, Mourad, Mounir, Nejm et Mehdi... à tous ceux que je vois encore et ceux que le vent a emmené vers d'autres rives.

Enfin, et surtout, je voudrais remercier ma famille, en particulier ma mère, mon père, mes deux frères, surtout mon frère Khaled mon meilleur soutien ces deux dernières années, ma sœur, ma tente Zohra et sa famille pour leurs soutiens et encouragements.

Table des matières

Avant-propos

All truths are easy to understand once they are discovered.
The point is to discover them. Galileo Galilei (1564 - 1642)

Chapitre 1

Introduction générale

1.1 Contexte de travail

De nos jours, l'utilisation croissante des systèmes de gestion de workflows dans les entreprises exprime leur importance indéniable comme outil d'automatisation de leurs procédés. Les systèmes de gestion de workflows[1] sont parmi les systèmes les plus élaborés pour définir et exécuter des procédés. Ils permettent, en particulier, de décrire explicitement les méthodes de travail réalisant un procédé, de les expérimenter, de mesurer leurs qualités et de les optimiser afin de pouvoir assurer l'amélioration et la réutilisation des procédés. Pendant la dernière décennie, les systèmes de gestion de workflows [JB96; Law97; Coa96; AM97; Ell99; vdAvH02a; Fis00; ABJ+04; YW03] se sont répondues offrant un fort potentiel de modélisation et d'automatisation. Si un procédé permet de décrire d'une manière informelle les méthodes de travail d'un groupe de personnes et les règles qui les régissent, un workflow permet de formaliser, de structurer, d'automatiser (dans la mesure du possible) et d'exécuter ces méthodes de travail.

En dépit de ce potentiel établi, les systèmes de gestion de workflows[2] montrent certaines limites. Les entreprises expriment un véritable besoin de systèmes de gestion de procédés évolués qui s'adaptent à la capitalisation de leurs systèmes d'information, à l'optimisation de leurs méthodes de production et l'évolution de leurs besoins et leurs méthodes de travail. Elles ressentent également un besoin de mécanismes de contrôle et d'outils de modélisation pour leurs systèmes de gestion de procédés afin de concevoir des modèles de workflow robustes assurant un traitement fiable et flexible des erreurs et des exceptions d'exécution.

Bien que les systèmes de workflows actuels soient établis par des modèles de procédés explicites, une des difficultés est née du fait que les procédés ne sont pas tous explicitement définis, du fait de la diversité des tâches et des intervenants (concepteurs, utilisateurs, gestionnaires, etc.) dans une application de conception/développement complexe et du caractère initialement imprévisible d'autres paramètres qui se manifestent après l'établissement du procédé (besoins des utilisateurs, échec ou exception d'exécution inattendues, etc.). En effet, dans le cadre d'une application complexe, où il faut parfois coordonner des fragments de procédés issus de modèles de procédés différents, d'environnement hétérogènes, et des intervenants issus d'organisations différentes, il est impossible de prévoir et de rendre compte initialement et facilement de tous les paramètres nécessaires à une conception «parfaite».

Par ailleurs, une évolution continue des procédés est le témoin, de nos jours, d'une très

[1] fluxgiciel est le terme français traduisant workflow mais il n'est pas usité

[2] Nous utiliserons interchangeablement les deux termes : procédés et workflows, pour décrire respectivement les modèles abstrait et opérationnel de définition et d'exécution des procédés métiers

grande diversification des services et des produits des entreprises. De nouveaux besoins émergent et les procédés existants changent («la seule constante est le changement»). Par conséquent, l'alignement des procédés aux évolutions observées exige une attention et une action continues. Pour «maintenir cet alignement» il est important de détecter les changements dans le temps, c.à.d, les déviations du comportement décrit ou prescrit. Des techniques de «**découverte de procédé**[3]» utilisant des traces d'exécutions pour découvrir le procédé «réel» sont des outils appropriés pour détecter ces déviations et les utiliser pour supporter l'évolution du procédé.

1.2 Objectifs de la thèse

L'objectif de recherche de cette thèse est d'assurer **une conception continue** qui répond aux problèmes de **fiabilité et d'évolution** des procédés métiers. Pour ce faire, il faut un **modèle de procédés correct et fiable** et des **mécanismes de conception «continus» et «réactifs»**. Par modèle de procédés correct, nous entendons un modèle de procédés dont sa conception respecte une sémantique assurant une construction syntaxique cohérente du schéma de workflow, ne contenant pas d'anomalies conceptuelles, et respectant le point de vue métier du procédé et les besoins des concepteurs. Par modèle de procédés fiable, nous entendons un modèle de procédés fournissant des mécanismes de recouvrement qui permettent la continuation du traitement de l'instance du workflow en cas d'échec d'exécution. Par mécanismes de conception «continus», nous entendons des mécanismes de conception dynamiques qui répondent aux besoins de re-ingénierie du procédé. Par mécanismes de conception «réactifs», nous entendons une méthodologie de conception qui intègre les nouvelles contraintes d'exécution et les besoins d'évolution d'un procédé.

Pour répondre au premier objectif d'un **modèle de procédés correct et fiable**, notre point de départ était de constater que les technologies de modélisation de procédés métiers actuelles reposent principalement sur deux concepts forts pré-existants : les systèmes de workflows spécifiant d'une part l'aspect coordination et organisationnel des procédés métiers, et les modèles transactionnels assurant d'autre part un certain niveau de correction des exécutions. Séparément ces deux modèles ne peuvent pas répondre aux besoins d'un modèle de procédé qui concernent à la fois les besoins métiers et les besoins de flexibilité et de fiabilité. En effet, bien que l'approche workflow soit une technique clé pour l'automatisation des procédés métiers, cette approche ignore souvent les problèmes liés à la fiabilité notamment en cas d'échecs. Réciproquement, bien que les modèles transactionnels aient su s'étendre en supportant des structures de contrôle plus complexes et en relâchant certaines des propriétés ACID (principalement l'atomicité et l'isolation), ces modèles restent très liés au contexte des bases de données traditionnelles. Ils montrent rapidement leurs limites lorsqu'ils sont appliqués au contexte de procédés métiers. Leurs limites découlent principalement de leur rigidité en termes de structure de contrôle et des contraintes qu'ils imposent aux concepteurs sans que ces derniers puissent les négocier.

Dans nos travaux, nous proposons de réconcilier ces deux approches en combinant la flexibilité métier des workflows et la fiabilité des modèles transactionnels. Concrètement nous proposons un modèle de workflow transactionnel spécifié formellement en utilisant le calcul événementiel. Ce modèle fusionne et allie d'une manière efficace les avantages de ces deux technologies en un seul modèle, en assurant des propriétés de flexibilité et d'expressivité. Pour ce faire, nous avons choisi d'enrichir la description des activités de workflow avec des propriétés transactionnelles pour mieux exprimer leurs comportements. Ensuite, nous avons développé notre modèle de workflow. Ce

[3]Les termes «découverte de procédé» et «découverte de workflow» sont interchangeables

modèle étend et enrichit le modèle classique du flux de contrôle[4] du workflow et le fusionne avec un modèle transactionnel. Le modèle transactionnel spécifie des dépendances transactionnelles intra et inter-activités décrivant un «flux transactionnel» décrivant, entre autres, les mécanismes de recouvrement en cas d'échec d'exécution. L'originalité de notre modèle de workflow transactionnel à ce niveau est la flexibilité que nous offrons aux concepteurs pour spécifier leurs besoins en termes de structure de contrôle (flux de contrôle) et de recouvrement (flux transactionnel). Ce modèle fera la base de la solution que nous allons proposer pour répondre au deuxième et principal objectif de la thèse qui propose des **mécanismes de conception «continus» et «réactifs»** des procédés métiers.

Pour répondre au deuxième objectif de **mécanismes de conception «continus» et «réactifs»**, nous avons d'abord relevé que les approches classiques de conception de procédés prennent généralement comme point de départ les perceptions et suppositions des concepteurs et souffrent d'une implication déficiente des utilisateurs tant dans la conception que la vérification. En plus, elles manquent de mécanismes de correction permettant d'optimiser la structure du workflow initialement conçu du fait qu'elle analyse un modèle figé et ne traite pas du besoin d'évolution pendant la phase d'exécution. Ainsi, notre défi est de relever le besoin d'évolution et d'adaptation du schéma du workflow, suite à des nouvelles contraintes d'exécution ou à des besoins d'évolution exprimés par l'utilisateur après l'établissement du procédé, par une nouvelle étape d'amélioration et/ou de correction dans le cadre d'une de re-conception continue du procédé.

Notre approche est d'inverser le processus de conception en prenant comme point de départ les modèles existants et les connaissances acquises suite à la phase d'exécution. Concrètement, nous proposons l'utilisation des techniques de «**découverte de procédé**» en spécifiant les méthodes qui permettent de découvrir et de comprendre les procédés à partir des traces d'exécutions en appliquant des techniques de fouille de données[5]. Le but est de créer une boucle rétro-active permettant de découvrir le workflow à partir de ses traces d'exécutions et par la suite de détecter des erreurs de la conception et d'adapter le modèle de workflow initial aux nouvelles contraintes découvertes. Notre approche pour la découverte et la re-conception dynamique et continue des workflows procède principalement en trois étapes. D'abord, nous découvrons le flux de contrôle des procédé. Après cela, nous spécifions, les mécanismes de recouvrement décrits par le flux transactionnel qui est lié au flux de contrôle par les règles sémantiques et syntaxiques de notre modèle de workflow transactionnel. Alors en se basant sur les résultats découverts, nous utilisons un ensemble de règles pour améliorer le traitement et le recouvrement des échecs d'exécution, et finalement la fiabilité de l'application. Ainsi cette approche nous permet à la fois d'avoir un outil d'aide à la re-conception pour mieux supporter l'évolution des procédés et de coller au mieux aux attentes «réelles» des utilisateurs par l'analyse de leurs traces d'exécutions. La faisabilité de notre approche est démontrée dans le système de gestion de workflows `BONITA` [MC03] et par notre outil de découverte de workflow `Workflowminer` [BGKM06] et intégré comme un plug-in dans la plate-forme générique de découverte de workflow `Prom` [vDdMV+05; GG06].

1.3 Organisation de la thèse

L'organisation de notre mémoire suit le schéma suivant.

Dans le chapitre 2, nous présentons le contexte de notre travail : l'ingénierie des procédés métiers. Nous montrons en particulier le cycle de vie des procédés et les avancées technologiques existantes pour pallier aux défis de cohérence et d'évolution de modèles. Nous soulevons les limites

[4] le terme anglophone est control flow

[5] Le terme anglophone est Data Mining

de ces propositions et le besoin d'un modèle fournissant des mécanismes dynamiques d'aide à la conception. Nous présentons ensuite la problématique de notre thèse : Quel modèle de procédés pour garantir une conception correcte, flexible et fiable ? et comment assurer une conception continue et réactive aux besoin d'évolution ? Nous motivons notre problématique via un exemple illustratif et nous synthétisons les limites des technologies actuelles pour résoudre ce problème. Enfin, nous présentons les principes directeurs de notre travail et les contributions de notre thèse.

Dans le chapitre 3, nous présentons un état de l'art décrivant les différents supports de modélisation. Nous étudions en particulier les propositions de workflows transactionnels par l'analyse de leurs structures conceptuelles et architecturales. Cette analyse nous permettra d'identifier les contours du modèle de procédés qui va répondre aux besoins de cohérence et de flexibilité. Par la suite, nous motivons les besoins des entreprises pour une conception continue et une exécution fiable de leurs procédés métiers en soulignant les limites des systèmes de gestion de procédés actuels à fournir une méthodologie de conception répondant aux besoins d'évolution et de fiabilité.

Les chapitres 4 et 5 constituent le cœur de notre travail. Nous détaillons respectivement la structure, les algorithmes et le mode opérationnel de notre approche de découverte et de re-ingénierie de procédé.

Dans le chapitre 4, nous dériverons, notre représentation de modèle de workflow transactionnel sous la forme d'un méta-modèle structuré en blocs de base (patrons de workflow). Nous développons ensuite cette structure de base afin d'inclure le flot transactionnel qui assure la gestion des erreurs potentielles d'exécution et les mécanismes de recouvrement. Nous introduisons la notion de workflow transactionnel et nous précisons comment modéliser son comportement selon des propriétés et des dépendances transactionnelles. Nous distinguons en particulier entre le flot de contrôle et le flot transactionnel d'un workflow.

Dans le chapitre 5, nous étudions la collecte des traces d'exécutions dans les systèmes de gestion de workflows en spécifiant les contraintes nécessaires pour mener notre processus de découverte de procédé. Par la suite, nous détaillons notre approche de découverte et de re-ingénierie de workflow, et les techniques relatives de recouvrement de workflows. Nous procédons principalement en trois étapes. D'abord, nous proposons un algorithme pour découvrir les patrons de workflow et les dépendances transactionnelles en utilisant une analyse statistique des traces d'exécutions. Ensuite, se basant sur les résultats découverts, nous utilisons un ensemble de règles pour améliorer le traitement et le recouvrement d'échecs d'exécution, et finalement la fiabilité des workflows. Nous illustrons l'applicabilité de chacune de ces étapes sur notre exemple de motivation.

Dans le chapitre 6, nous nous sommes intéressés à la validation de notre approche à travers le développement et l'utilisation d'un certain nombre d'outils et d'applications. Finalement, le chapitre 7 résumera nos contributions et dégagera les perspectives directes de nos travaux de recherche.

Chapitre 2

Contexte et problématique

Table des matières

2.1 Contexte de la thèse : l'ingénierie des procédés métiers

2.1.1 Gestion des procédés métiers

Le terme «la gestion des procédés métiers (BPM)» est un terme populaire aussi bien dans le domaine métier que scientifique qui relève des questions qui concernent, notamment la conception, l'analyse, la modélisation, l'exécution et le contrôle des procédés métiers [Sch96; DK00; SM01; vdAvH02b]. Le concept de procédés métiers a été défini comme «un ensemble d'activités logiquement liées pour réaliser un procédé métier prédéfini» [DS90]. Nous n'essayerons pas d'étendre ou de raffiner cette définition d'autant que d'autres définitions ont été proposées, mais de s'intéresser aux ingrédients remarquables de la gestion du procédé métier relatives, en particulier, aux méthodes, techniques, et outils supportant la conception, l'exécution, et l'analyse des procédés métiers opérationnels ou workflows.

Nous adoptons la même approche procédés métiers que Leymann et autres [LA94] qui distinguent deux aspects fondamentaux, à savoir l'aspect temps de construction et l'aspect exécution. L'aspect temps de construction se concentre sur la conception du procédé métier ; l'aspect exécution se centre sur le contrôle de l'exécution et le traitement des échecs d'exécution. Hammer et Champy disent essentiellement la même chose [HC93], mais ils soulignent également l'aspect

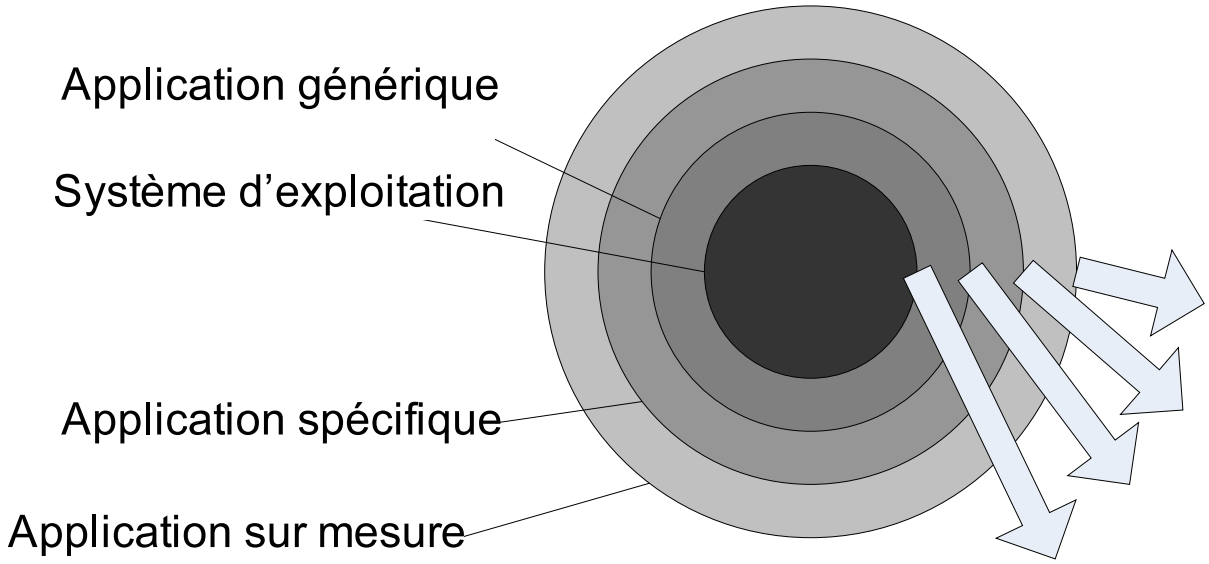
centré client d'un procédé métier qui est «une collection d'activités qui prend un ou plusieurs types d'entrées et fournit un résultat qui est de valeur pour l'utilisateur» . L'aspect centré client d'un procédé se concentre sur le rôle central des utilisateurs dans la validation de l'efficacité du procédé. En utilisant cette distinction nous considérons BPM comme le domaine de conception et d'exécution des procédés métiers en vue de satisfaire les besoins métiers de l'utilisateur. La distinction de ces trois concepts est également devenue commune au domaine des systèmes de gestion de workflows pour discuter leurs fonctionnalités principales [JB96].

Ces trois dimensions sont le cœur des motivations et des propositions de notre thèse. La première dimension qui est la conception du procédé métier est traditionnellement vue comme une issue stratégique et se concentre typiquement sur les aspects de la méthodologie de conception. La deuxième dimension qui est le support de l'exécution du procédé métier s'intéresse plus aux niveaux des décisions qui sont prises en temps d'exécution et notamment les décisions concernant le traitement des échecs d'exécution. Enfin, la troisième dimension qui est l'utilisateur du procédé métier s'intéresse à l'intégration des besoins métiers de l'utilisateur tout au long du cycle de vie du procédé.

2.1.2 Évolution des systèmes d'information d'entreprises

Pour montrer l'importance des systèmes de gestion de procédés métiers, il est intéressant de les mettre dans une perspective historique. La figure 2.1 montre que les systèmes d'information se composent d'un certain nombre de couches [vdAtHW03]. Le centre est constitué par le système d'exploitation. La deuxième couche se compose des applications génériques utilisées par un grand nombre d'entreprises. D'ailleurs, ces applications sont typiquement utilisées dans des départements multiples au sein de la même entreprise (par exemple, système de gestion de base de données, éditeur de texte, tableurs, etc.). La troisième couche se compose des applications spécifiques au domaine. Ces applications sont seulement utilisées dans des types spécifiques d'entreprises et de départements. La quatrième couche se compose des applications sur mesure. Ces applications sont développées pour des organisations spécifiques.

Figure 2.1 Système d'information pour les entreprises



Les tendances représentées dans la figure 2.1 fournissent un contexte historique pour les systèmes de gestion de procédés métiers. En effet, les systèmes de gestion de procédés métiers sont des applications séparées résidant dans la deuxième couche. Les exemples les plus remarquables des systèmes de gestion de procédé métiers résidant dans la deuxième couche sont les systèmes de gestion de workflows comme Staffware, MQSeries, COSA et FLOWer [vdAvH02a; JB96; Law97; LR00; Mar02]. Notons que les principaux systèmes de planification de ressources d'entreprises peuplant la troisième couche offrent également un module de gestion de workflows. Les moteurs de workflows de SAP, Baan, PeopleSoft, Oracle, et JD Edwards peuvent être considérés comme des systèmes intégrés de gestion de procédé métiers.

Dans les années 60, la deuxième et la troisième couche étaient absentes ou oubliées. Les systèmes d'information ont été établis au dessus de petits systèmes d'exploitation avec une fonctionnalité limitée. Comme aucun logiciel générique ni spécifique au domaine n'était disponible, ces systèmes sont principalement composés d'applications sur mesure. Dés lors, la deuxième et la troisième couche se sont développées et la tendance courante est que les quatre cercles augmentent de taille, c.à.d, elles se déplacent vers l'extérieur en absorbant de nouvelles fonctionnalités métiers.

Dans les années 70 et 80, des travaux comme ceux de Ellis [Ell79] et Holt [Hol86] ont déjà proposé des systèmes d'information motivés par des modèles de procédés métiers explicites. Pendant les années 70 et les années 80 il y avait un grand optimisme au sujet de l'applicabilité de ces systèmes. Malheureusement, peu d'applications ont réussi. En raison de ces expériences, l'application de cette technologie et les recherches scientifiques liées se sont presque arrêtées pendant une décennie. En conséquence, peu d'avancées scientifiques ou technologiques ont été observées dans les années 80. Dans les années 90, un regain d'intérêt énorme pour ces systèmes s'est manifesté. Le nombre de systèmes de gestion de workflows développés dans la dernière décennie et les nombreux articles sur la technologie de workflows illustrent la renaissance des systèmes d'information dédiés aux procédés métiers.

Les échecs dans les années 80 peuvent s'expliquer par des problèmes techniques et conceptuels. Dans les années 80, les réseaux étaient lents ou inexistants, il n'y avait aucune interface graphique appropriée, et tout logiciel de développement approprié était absent. Cependant, il y avait également des problèmes plus fondamentaux comme l'absence d'**une méthode unifiée de modélisation** de procédés. La plupart des problèmes techniques ont été résolus à ce jour. Cependant, **des problèmes plus conceptuels** demeurent non résolus.

Un des problèmes est que ces systèmes exigent une conception de workflow, c.à.d, qu'un concepteur doit construire un modèle détaillé décrivant exactement le déroulement du travail. La conception des workflows est loin d'être insignifiante : elle exige une connaissance profonde du procédé métier réel et des paramètres d'évolution au cours du temps. Les besoins d'évolution et d'adaptation aux nouveaux contextes peuvent se manifester particulièrement pendant l'exécution et influencer l'efficacité et la fiabilité du procédé métier.

2.1.3 Cycle de vie des systèmes de gestion de workflows

Comme nous l'avons indiqué dans la section précédente, la gestion des procédés métiers inclut des méthodes, des techniques, et des outils pour supporter la conception, l'exécution, la gestion, et l'analyse des procédés métiers opérationnels[6]. Elle peut être considérée comme une extension des systèmes et des approches classiques de gestion de workflows (WFM).

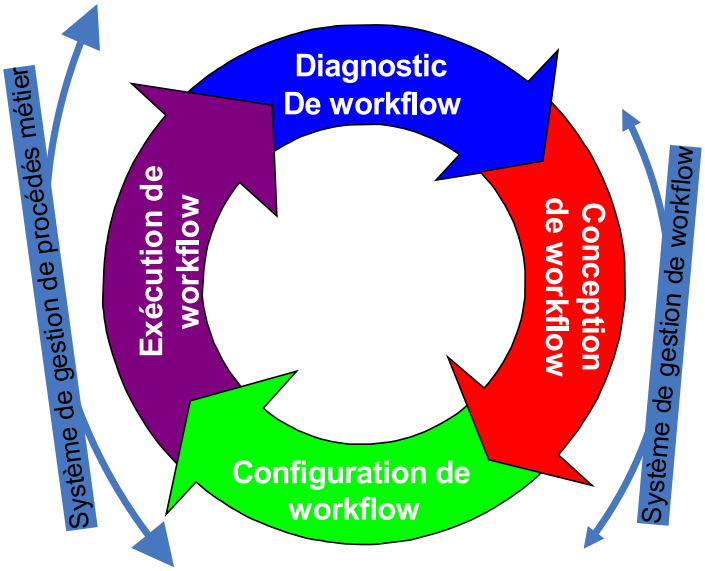
Le schéma de la figure 2.2 montre la relation entre WFM et BPM en utilisant le cycle de vie de BPM. Le cycle de vie de BPM décrit les différentes phases de support des procédés métiers.

[6]le terme opérationnel indique que ces procédés métiers ont été implantés dans un système de gestion de workflows

Dans la phase de conception, les procédés sont (re)conçus. Dans la phase de configuration, les conceptions sont mises en application en configurant le système d'information (par exemple, un sustème de gestion de workflows) pour implanter le procédé métier. Après la configuration, la phase d'exécution, où les procédés métiers sont exécutés en utilisant le système configuré, commence. Dans la phase de diagnostic, les procédés sont analysés pour calculer des différentes mesures de performance évaluant le procédé exécuté.

Comme le schéma de la figure 2.2 le montre, l'intérêt traditionnel d'un système de gestion de workflows se centre sur la moitié inférieure (c.à.d les phases de conception, de configuration et d'exécution) du cycle de vie de BPM [vdAvH02b]. Par conséquence, il y a peu de support pour la phase de diagnostic. En plus, le support de la phase de conception est limité à fournir un éditeur, alors qu'une analyse et une vraie aide à la conception sont absentes. Il est remarquable que peu de systèmes de gestion de workflows soutiennent le support, la simulation, la vérification, et la validation de la phase de conception de procédés. Aussi bien, il est également remarquable que peu de systèmes soutiennent l'interprétation des données d'exécution en temps réel. Notons que la plupart de systèmes de gestion de workflows collectent les traces d'exécutions des instances exécutées de procédés. Cependant, aucun outil pour supporter une une conception se basant sur cette source importante d'information n'est offert par les systèmes de gestion de workflows actuels [vdA05].

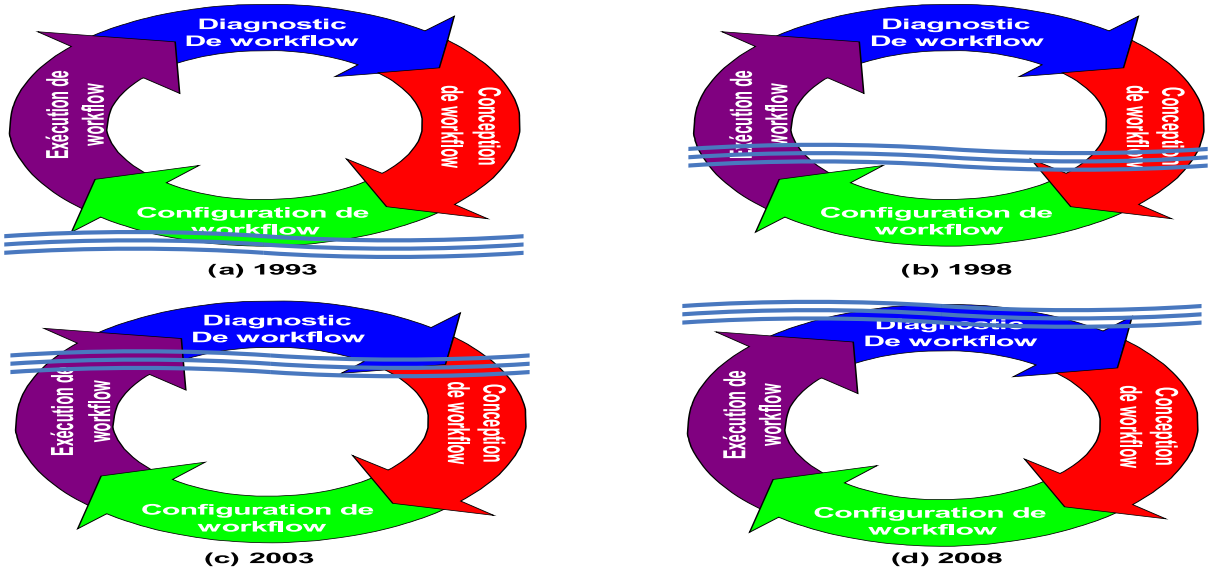
Figure 2.2 Cycle de vie de workflow et cycle de vie BPM



Inconstablement, la conception des procédés métiers est le secteur du BPM qui a suscité une très grande attention pendant les deux dernières décennies. La manière dont les procédés métiers sont structurés a un grand impact sur le coût, et la qualité de leurs produits. Les termes tels que BAM (*Business Activity Monitoring*), BOM (*Business Operations Management*), BPI (*Business Process Intelligence*) illustrent l'intérêt pour la fermeture de la boucle du cycle de vie de workflow. Ceci est illustré par la figure 2.3 qui montre le niveau de support sur quatre années différentes [vdAtHW03]. Les systèmes de gestion de workflows ont commencé par assurer des outils de support et d'automatisation des procédés métiers (c.à.d, la phase de configuration du système). Depuis les années 90, plus d'emphases a été mise pour supporter et améliorer la phase de conception. Maintenant de plus en plus de fournisseurs essayent de clôturer le cycle de vie de

BPM en ajoutant la fonctionnalité de diagnostic et de re-ingénierie. Les termes à la mode tels que BAM, BOM, BPI, etc. illustrent ces tentatives.

Figure 2.3 Le niveau de support du cycle de vie BPM



Actuellement, beaucoup de fournisseurs de systèmes de gestion de procédés sont en train de repositionner leur produit. L'analyse des systèmes de gestion de procédés (BPA)[7] devient un aspect important dans le processus de développement et de croissance des systèmes de gestion de procédés. Notons que BPA couvre des aspects souvent ignorés par les systèmes de gestion de workflows classiques (par exemple, diagnostic, simulation, etc.). Par ailleurs, il existe plusieurs mécanismes de contrôle actifs de procédés, également appelés mécanismes de surveillance (monitoring) de workflows, qui analysent des instances de workflows au cours d'exécution. La surveillance active de l'état actuel des instances de workflows peut répondre à plusieurs objectifs, tels que la génération de rapports d'exceptions, de retard de traitement ou de rapports de détection précoces pour les activités potentiellement en retard de traitement. Elle peut fournir des informations sur l'état des instances courantes de workflows, par exemple pour répondre à une enquête d'un utilisateur au sujet de l'état d'une instance. Cependant elle ne fournit pas d'outils d'amélioration ou de re-ingénierie du modèle de workflow.

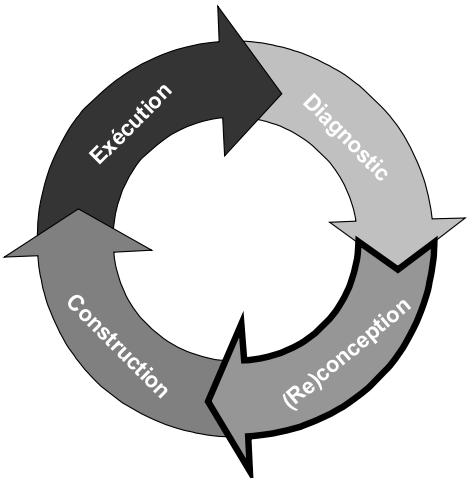
2.1.4 Ré-ingénierie des procédés

Des outils et des techniques de conception des procédés métiers sont souvent présentés comme «intelligents» ou «avancés», bien qu'ils ne conçoivent pas «activement» les procédés métiers. En effet, très peu d'outils proposent des mécanismes d'amélioration et de correction conceptuelle. Hammer [Ham90] et Davenport et Short [DS90] étaient les premiers à décrire des approches plus ou moins systématiques pour une amélioration et une refondation entière et radicale de l'ingénierie des procédés métiers par reconstruction radicale de la conception. Cette approche est désignée avec les termes de «re-ingénierie de procédés métiers» par Hammer [Ham90] ou de

[7] le terme anglophone est Business Process Analysis

«re-conception de procédés métiers» par Davenport et Short [DS90]. Ces idées regroupées sous l'acronyme BPR (Reconfiguration de processus)[8] ont été largement adoptées par l'industrie.

Figure 2.4 La re-ingénierie des procédés métiers



BPR suit un cycle de vie (*c.f.* la figure 2.4) qui commence par une initiative, venant la plupart du temps de l'équipe de direction et comporte un certain nombre de phases :

1. La phase de diagnostic commence par une analyse de la situation actuelle, et en particulier des problèmes provoqués par les méthodes de travail existantes. Entre autres, elle montre où les méthodes de travail existantes ne produisent pas le résultat désiré.
2. Une fois que le diagnostic a été fait, la phase de (re)conception suit. Les méthodes de travail existantes ne seront pas utilisées comme base de travail : une nouvelle description du procédé est produite.
3. Dans la phase de (re)construction, le nouveau procédé est implanté dans un système de définition de procédés (un système de gestion de workflows par exemple).
4. Pendant la phase opérationnelle, la performance des procédés est mesurée et évaluée en utilisant des critères prédéfinis. Ceux-ci peuvent bien justifier le lancement d'un nouveau cycle de re-ingénierie.

La popularité de la reingénierie des procédés dans l'industrie s'est développée considérablement depuis son introduction. Une compagnie de services australienne a réalisé un sondage auprès de 107 compagnies australiennes et asiatiques et a rapporté que 50% étaient déjà intéressées ou projettent de l'être par la re-ingénierie des procédés. Dans une étude des compagnies américaines et européennes en 1994, 621 compagnies américaines et européennes avec des revenus de 500 millions de dollars au moins par an ont été auditionnées. Plus de 69% de ces compagnies avaient déjà adopté la re-ingénierie des procédés en tant que moyens d'améliorer leurs opérations commerciales. Aussi bien que 88% des compagnies américaines utilisaient la re-ingénierie des procédés ou étaient sur le point de lancer des projets de reingénierie des procédés. Une étude britannique semblable a rapporté un pourcentage de 59%. Une autre étude, qui a inclus 93 grandes et moyennes compagnies finlandaises, a prouvé que 41% de ces compagnies ont conduit un ou plusieurs projets de re-ingénierie des procédés. Ceux-ci et beaucoup d'autres études semblent suggérer que la re-ingénierie des procédés soit plus populaire parmi de plus grandes compagnies.

[8] le terme anglophone est Business process reengineering

Par ailleurs, on suspecte qu'à la différence des compagnies nord américaines les compagnies dans d'autres régions du monde (par exemple, l'Europe) sont beaucoup plus lentes à adopter cette approche [Zam94] .

2.2 Problématique de la thèse : Comment assurer une exécution fiable et une conception continue ?

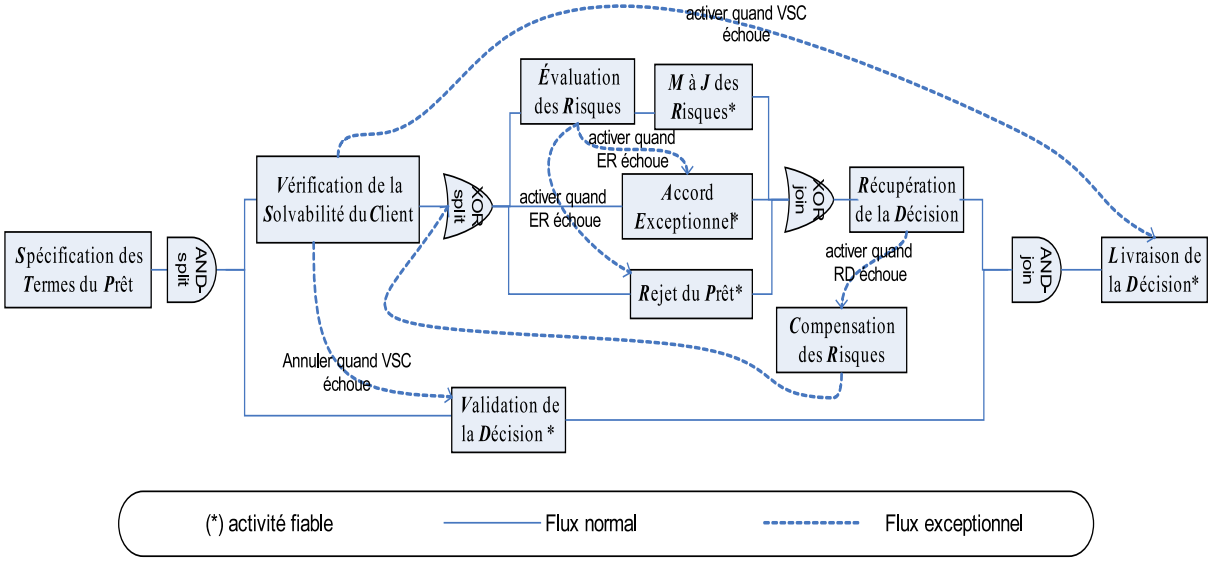
Concevoir un workflow est un processus long et compliqué qui exige une connaissance profonde des paramètres, contraintes et besoins métier du procédé impliquant beaucoup de discussions entre intervenants. En effet, la conception de workflows exige un modèle détaillé et précis décrivant exactement le déroulement du travail. Les technologies actuelles de conception de procédés traitent en amont la phase de conception proposant ainsi une approche «simpliste» qui se base sur des suppositions à priori et est souvent incapable de ce fait de répondre efficacement aux besoins d'évolution. Cette conception fournit souvent un modèle prescriptif, contenant ce qui «doit» être fait, plutôt que de décrire le procédé réel. En conséquence, les modèles tendent à être subjectifs, ce qui implique des anomalies entre les schémas de workflow initialement conçus et les procédés perçus par ceux qui l'ont conçu ou l'utilisent. En conséquence, les utilisateurs peuvent dévier de la conception pré-spécifiée.

Exemple de motivation

Considérons un exemple simplifié de traitement de demande de crédit bancaire. Le workflow de traitement de demande de prêt est représenté sous forme graphique dans la figure 2.5. D'abord, le client spécifie la quantité et les termes du prêt via l'activité de «**S**pécification des **T**ermes du **P**rêt (STP)». Alors une instance de workflow est lancée pour la collecte d'information sur le client et évalue son degré de solvabilité par l'activité de «**V**érification de la **S**olvablité du **C**lient (VSC)». Après cela, la banque prend sa décision en choisissant exclusivement entre trois options : (i) Elle évalue les risques pour la banque via l'activité «**É**valuation des **R**isques (ER)» et met à jour la valeur de sa prise de risque globale «**M**ise à **J**our du **R**isque (MJR)» ou (ii) le client est suffisamment important pour lui accorder directement le prêt, sans évaluation de risque. Dans ces circonstances un cadre dirigeant du service de crédits devrait approuver le prêt «**A**ccord **E**xceptionnnelle (AE)» ou (iii) on rejette le prêt par l'activité «**R**ejet du **P**rêt (RP)». Par la suite, l'activité de «**R**écupération de la **D**écsion (RD)» rédige et enregistre la décision d'acceptation ou de rejet de la demande dans la base de données de la banque. La livraison de la décision au client (contrat de prêt ou avis de rejet) par l'activité «**L**ivraison de la **D**écison (LD)» ne peut pas être faite sans approbation d'un contrôleur (direction de la banque). En effet, une activité humaine «**V**alidation de la **D**écison (VD)» est présente tout le long du procédé de décision pour contrôler ses activités. Ainsi, on peut seulement accorder le prêt si la direction de la banque le valide. Elle peut rejeter la demande de prêt même si une décision positive a été prise. Le contrôleur peut donner librement sa décision à tout moment pendant le processus de décision.

Initialement, pour traiter les échecs d'exécution des activités composantes du workflow, les concepteurs spécifient en plus du `flot de contrôle` des mécanismes de recouvrement qui décrivent des interactions transactionnelles additionnelles entre les activités (flèche pointillé). Dans notre exemple, on a indiqué que si RD échoue alors l'activité «**C**ompensation des **R**isques (CR)» compense les effets de cet échec et le procédé de décision de prêt devrait être re-exécuté. En plus, en cas d'échec de ER, le workflow continue l'exécution par un accord exceptionnel du prêt AE

Figure 2.5 Workflow exemple de prêt bancaire.



ou une décision de rejet du prêt RP. Par ailleurs, si LD échoue elle sera (re)exécutée jusqu'au succès. Quant à l'échec de VSC, le workflow termine son exécution par l'activité LD tout en abandonnant l'exécution de VD (la demande de prêt est rejetée). Finalement, des mécanismes de traitement d'échecs n'ont pas été fournis pour les autres activités qui sont supposées fiables (exemptés de tout échec).

Le problème majeur, auquel nous nous intéressons, est comment garantir des exécutions d'instances de workflow fiables, c.à.d que les mécanismes de recouvrement spécifiés sont suffisants pour pallier à tout échec d'exécution et que la conception initiale du workflow va répondre aux besoins d'évolution des utilisateurs qui se manifesteraient dans la phase d'exécution. Par exemple, certaines parties (activités ou mécanismes de recouvrement) du procédé peuvent *ne jamais être atteintes* lors de la phase d'exécution ou des *nouvelles* contraintes peuvent être observées et omises dans la phase de conception initiale (échec d'une activité pré-spécifiée comme fiable). Ainsi des erreurs de conception peuvent se produire et devraient être corrigées.

Concrètement, supposons par exemple qu'en réalité (par l'observation d'un nombre suffisant d'instances d'exécution) VSC n'échoue jamais mais VD peut échouer. Ce constat, nous mène à entreprendre une série de corrections et d'ajustements pour répondre d'abord aux nouveaux besoins des utilisateurs découverts dans la phase d'exécution tout en assurant un modèle de workflow correct, fiable et optimal. Par exemple, nous pouvons conclure à partir des faits découverts, que d'une part qu'il n'y a aucun besoin de spécifier un mécanisme de recouvrement pour VSC et ainsi suggérer de supprimer ce mécanisme de recouvrement qui peut être cher, et d'autre part suggérer d'avorter les activités concourantes du processus de décision et reprendre l'exécution du workflow comme mécanisme de recouvrement à l'échec de VD.

Le but ambitieux de notre travail est celui de créer une base pour un cadre de modélisation capable à la fois d'assurer une conception «cohérente» et «flexible» et un outil d'aide à la conception permettant une conception «intelligente» et «continue». L'idée est d'inverser le processus de conception et de rassembler des données au cours d'exécution, pour supporter la conception et l'analyse de procédés. L'information est collectée au temps d'exécution, habituellement enre-

gistrée dans des traces d'exécutions et utilisée pour découvrir le modèle de workflow réellement exécuté. Nous appelons cette activité «découverte de procédé».

Ainsi, c'est une question d'opportunité et de besoin : l'opportunité est fournie par le fait que pratiquement chaque système de gestion de procédés collecte des traces d'exécutions qui peuvent alors être utilisées pour une analyse à posteriori du comportement du procédé à l'exécution. Ces traces d'exécutions sont utilisées actuellement pour l'évaluation et la surveillance des procédés ainsi qu'à l'analyse de leur impact économique à travers le calcul d'indicateurs de performances. Cependant, elles peuvent fournir une source valable pour améliorer la conception et la fiabilité des procédés. Le besoin est occasionné par la nécessite d'avoir une connaissance plus profonde des procédés à l'exécution pour mieux supporter leur évolution ou corriger leurs anomalies. Ce but est à double usage : développer des théories et des techniques qui expliquent, découvrent et comprennent le comportement du procédé à l'exécution et avoir une base solide pour développer des systèmes qui supportent efficacement une (re)conception continue, assurent et contrôlent une exécution fiable des procédés. Pour ce faire notre travail doit répondre aux interrogations suivantes :

1. Comment spécifier et identifier les propriétés conceptuelles du modèle de procédés qui sont corrélées aux problèmes d'évolution et de fiabilité ?
2. Quelles contraintes sur la structure et la richesse des données de traces d'exécutions sont nécessaires au processus de découverte de procédé ?
3. Comment les techniques de fouille de données peuvent-elles être utilisées pour la découverte de procédé ?
4. Quel outil utiliser pour l'analyse des mécanismes de recouvrement et l'amélioration de la fiabilité d'exécution du procédé ?
5. Comment utiliser les résultats du processus de découverte pour améliorer le traitement des échecs d'exécution et optimiser la fiabilité et le coût des mécanismes de recouvrement dans le cadre du processus de re-ingénierie des procédés ?

2.3 Principes, approche et contributions de la thèse

2.3.1 Principes et outils directeurs

Notre premier objectif est donc de proposer une approche assurant une modélisation fiable de workflows garantissant des exécutions correctes à travers des mécanismes de recouvrement sémantiquement corrects. Compte tenu du contexte de notre travail et des besoins décrits ci-dessus, nous avons identifié trois dimensions à prendre en compte qui représentent les principes directeurs de notre modèle de workflow :

- **Flexibilité** : Il faut laisser la liberté aux concepteurs de spécifier leurs besoins tant au niveau de la structure de contrôle de l'application qu'au niveau du degré de correction exigé. Ceci sous entend une flexibilité du modèle de workflow pour pouvoir à la fois supporter des structures de contrôle assez complexes et fournir des outils de modélisation aux concepteurs pour spécifier leurs critères de correction.
- **Correction** : Il faut pouvoir assurer des exécutions correctes et cohérentes qui réagissent au(x) échec(s) conformément aux besoins des concepteurs du workflow. Ceci implique le développement d'un ensemble de règles sémantiques et syntaxiques qui assurent une cohérence du schéma de workflow en évitant certaines erreurs conceptuelles de types : interblocage, circuits absorbants, activités non atteintes, flots erronés, etc.

- **Besoins des procédés métiers** : il faut que l'ensemble des techniques développées et sémantiques utilisées soit approprié aux caractéristiques des procédés métiers. Par exemple, annuler le travail effectué si le critère de correction n'est pas vérifié n'est pas adéquat au contexte des procédés métiers. En effet, d'une part il n'est pas pratique d'annuler un travail qui a duré des jours voir des mois. D'autre part, il n'est pas toujours évident d'annuler le travail effectué dans le contexte de procédés métiers. Donc, notre modèle doit fournir des structures de contrôle et de correction adaptées au contexte métier.

Un dernier principe directeur, qui est commun à toute approche, est de garder le maximum de simplicité possible tant au niveau des concepts utilisés qu'au niveau de leur utilisation.

Notre deuxième objectif est de fournir mécanismes de conception «continus» et «réactifs» aux besoins d'évolution dans une démarche de re-ingénierie des procédés métiers. Pour ce faire nous adoptons une méthodologie de conception «intelligente» et «continue» qui apprend par l'analyse des traces d'exécutions. Notre méthodologie de conception peut être assimilé à des termes issus du domaine de l'apprentissage automatique tels que : «pour gagner de la connaissance, ou la compréhension, par étude, instruction, ou expérience» [Mit95].

Les concepteurs humains produisent souvent des procédés qui ne fonctionnent pas comme voulu. En fait, certaines caractéristiques pourraient ne pas être complètement connues à la phase de conception ou peuvent évoluer dans la phase d'exécution. La «découverte de procédé» par l'analyse des traces d'exécutions qui se base sur des techniques de fouille de données peut être utilisée pour découvrir des relations et des corrélations importantes qui restent difficile à décoder ou invisibles pour le concepteur humain. Ainsi, nous adoptons une méthode inductive qui se base sur des observations du monde réel. Dans la méthode inductive, les observations du monde réel sont l'autorité. Si un raisonnement est en conflit avec ce qui se produit dans le monde réel (c.à.d la phase d'exécution), le raisonnement doit être changé ou abandonné. L'application de la méthode inductive se fait par un processus d'inférence inductif qui peut être défini comme «processus d'extraction de règles générales à partir d'exemples» ([AS83]). Dans cette thèse nous utilisons ce processus, c.à.d nous nous servons de l'inférence inductive pour induire et développer les modèles à partir des données collectées à l'exécution. Et si le modèle découvert contredit le modèle pré-conçu, l'autorité revient au modèle découvert.

2.3.2 Approche

Afin d'atteindre notre premier objectif tout en prenant en compte les principes directeurs que nous nous sommes fixés ci-dessus, nous avons adopté une démarche qui repose sur les points suivants :

- Enrichir la description des activités du workflow pour mieux exprimer leurs propriétés transactionnelles et développer un modèle qui fusionne la fiabilité des modèles transactionnels et la flexibilité des systèmes de workflows. Ce modèle permet la spécification d'un modèle de workflow transactionnel, bien formalisé et développé en calcul événementiel. Il sert de base à l'ensemble des techniques de découverte utilisées par la suite. Dans notre approche, nous distinguons le `flot de contrôle` et le `flot transactionnel` d'un workflow. Le `flot de contrôle` décrit l'ordre d'exécution des activités. Le `flot transactionnel` définit les mécanismes de gestion d'échecs et de recouvrement.
- Fournir un moyen aux concepteurs pour exprimer leurs besoins de fiabilité en terme de semi atomicité[9] à travers la définition de mécanismes de recouvrement qui seront flexibles

[9]Le terme semi atomicité est notre traduction de l'anglicisme «failure atomicity». La semi atomicité est la relaxation de la propriété d'atomicité du modèle ACID. Elle fait référence à des exécutions correctes même en présence d'échecs de certaines sous-transactions.

et relatifs à chaque activité. Ces mécanismes se basent principalement sur la spécification par le concepteur d'un «point cohérent» pour chaque activité d'où on peut reprendre l'exécution de l'instance de workflow [DDS97].
- Développer des règles liant le `flot de contrôle` et le `flot transactionnel` et assurant une cohérence sémantique et une correction syntaxique du schéma de workflow.

Afin d'atteindre notre deuxième objectif, nous sommes intéressés, par la suite, à induire les modèles de procédé en se basant sur les traces d'exécutions. Ainsi, tout en prenant en compte les principes directeurs que nous nous sommes fixés ci-dessus, nous avons adopté une démarche qui repose sur les points suivants :
- Proposer des outils de collecte de traces d'exécutions, définir un format unique de traces d'exécutions, spécifier et démontrer la notion de traces d'exécutions «complétes» qui garantit une équivalence entre le modèle découvert et la réalité.
- Développer des outils mathématiques d'analyse statistique de traces d'exécutions pour extraire des corrélations d'exécution utilisées par le processus de découverte de procédé.
- Spécifier statistiquement les différents types de dépendances du `flot de contrôle` et du `flot transactionnel` sous la forme de règles et prouver une correspondance mutuelle entre les règles statistiques et le comportement effectif. Ces règles sont utilisées pour découvrir les patrons de workflow décrivant le `flot de contrôle` et les propriétés et dépendances transactionnelles décrivant le `flot transactionnel`.
- Proposer des règles de re-conception permettant la correction et l'amélioration du schéma de workflow initialement conçu. Ces règles se basent sur les disparités détectées entre le modèle initialement conçu et le modèle découvert et respectent à la fois les règles sémantiques et syntaxiques assurant un schéma de workflow cohérent et correct ainsi que le respect des besoins métiers du procédé exprimés par les concepteurs.
- Pour examiner la capacité de notre méthode à surmonter les difficultés croissantes de passage à l'échelle, nous développons des données d'exécution simulées pour nos test de validation. L'avantage d'exécuter des expériences utilisant la simulation est qu'il est plus facile de contrôler des variables externes, assurant une meilleure validité interne.

2.3.3 Contributions

Les contributions de notre thèse sont :
- Un outil de conception, sous la forme d'un modèle de workflow transactionnel, permettant une conception fiable et cohérente et établissant un ensemble de règles sémantiques assurant des mécanismes de recouvrement fiables et corrects. Il fusionne les systèmes de workflows et les modèles transactionnels, deux technologies fortes, souvent considérées concurrentes. Notre modèle révèle à la fois un aspect transactionnel et un aspect coordination. Par rapport aux modèles transactionnels, notre modèle permet des structures de contrôle plus complexes. Par comparaison aux systèmes de workflows classiques, notre modèle intègre un comportement transactionnel, une dimension relativement ignorée. En plus, l'utilisation des patrons comme concept de base dans notre modèle de workflow assure un haut niveau d'abstraction et de simplicité. Par ailleurs, ce modèle ne se limite pas à spécifier le schéma de workflow, mais il propose aussi un certain nombre de règles de cohérence sémantique et syntaxique.
- Un moyen aux concepteurs pour exprimer leurs besoins de fiabilité en terme de semi-atomicité. Contrairement aux modèles transactionnels classiques, nous suivons le sens inverse en partant des spécifications des concepteurs pour définir les mécanismes de recouvrement nécessaires pour assurer la fiabilité selon leurs besoins métiers. Pour ce faire, nous

utilisons dans notre approche, la notion de «point de cohérence». Comme son nom l'indique, ce point spécifie pour chaque activité échouée, le point à partir duquel l'instance du procédé doit recouvrir l'exécution correctement. La localisation de ce «point de cohérence» dépend à la fois des règles sémantiques et syntaxiques du procédé et des choix métiers du concepteur.

- Une nouvelle technique de découverte du `flot de contrôle` du workflow par la découverte de ses patrons. L'originalité de cette nouvelle approche s'explique par l'algorithme adopté qui se base sur la combinaison de deux méthodes complémentaires, l'une statistique, l'autre purement algorithmique. Ceci a apporté de nouveaux mécanismes qui permettent l'analyse de traces d'exécutions incomplètes. Nous avons pu aussi améliorer la découverte de la concurrence en proposant un mécanisme de «fenêtrage» lors de l'analyse des traces d'exécutions, et spécifier la nature des points de «jointure» et de «diffusion» à travers des patrons de workflow. Nous avons aussi défini les contraintes sur la qualité et la quantité des traces d'exécutions nécessaires aux processus de découverte de procédé.
- La découverte du comportement transactionnel du workflow par l'analyse des instances contenant des échecs d'exécution. Nous y distinguons deux types de dépendances transactionnelles : les dépendances transactionnelles intra-activité qui permettent de découvrir les propriétés transactionnelles des activités et les dépendances transactionnelles inter-activités qui permettent de découvrir le flux transactionnel. Ces dépendances transactionnelles spécifient finalement les mécanismes de recouvrement découverts.
- Des règles pour corriger ou améliorer les techniques de recouvrement dans le cas où elles ne répondraient pas aux besoins des utilisateurs. Ces règles suivent les règles syntaxiques et sémantiques de cohérence de notre modèle de workflow transactionnel mais aussi les recommandations conceptuelles des utilisateurs. Ainsi, le schéma de workflow spécifié en amont avant l'exécution peut être sujet à un ajustement selon le schéma découvert.
- L'implantation d'un outil de collecte de traces d'exécutions et l'instantiation des scénarios de workflows dans le système de gestion de workflows Bonita, la validation des techniques de découverte de procédé proposées et la réalisation de tests d'exécution de validation sur un panel de traces d'exécutions simulées ou collectées par notre outil de découverte de workflow `Workflowminer` [BGKM06], qui a été intégré aussi comme un plug-in dans la plate-forme générique de découverte de workflow `ProM` [vDdMV+05; GG06].

2.4 Conclusion

Aujourd'hui, des systèmes de gestion de workflows sont mis en oeuvre dans plusieurs organisations en tant qu'outils de gestion de procédés métiers (BPM). Plus les procédés deviennent automatisés, plus le centre d'intérêt de l'industrie et des milieux universitaires se décale du déploiement à la surveillance, l'analyse, et l'optimisation de procédés. L'installation et le maintien d'un système de gestion de procédés exigent des modèles qui prescrivent comment les procédés devraient être contrôlés et gérés par le système. Les propositions actuelles de gestion de workflows n'offrent pas de solutions efficaces assurant à la fois une modélisation garantissant une **exécution fiable** et **une conception continue**. D'une part, les systèmes de gestion de workflows actuels ignorent souvent la dimension de correction dans les outils de modélisation qu'ils proposent, ce qui peut engendrer des modèles de procédés peu fiables à l'exécution, faute de mécanismes de recouvrement adéquats. D'autre part, le cycle de vie des procédés soufre actuellement d'un déséquilibre qui se manifeste par une phase diagnostic quasi-absente dans le cycle de vie des workflows existants.

Avec la distribution des facilités de traitement de l'information aux entreprises, les enjeux pour des systèmes de gestion de procédés augmentent. Les données des traces d'exécutions d'instances de workflows fournissent une source valable pour l'analyse des systèmes de gestion de workflows. En plus, la technologie de workflows évolue dans une direction de flexibilité plus opérationnelle pour traiter de l'évolution et le traitement d'échecs d'exécution des workflows. Découvrir et exploiter les déviations entre le schéma de workflow pré-conçu et le schéma de workflow exécuté nous permet de corriger ces déviations ou/et de mieux comprendre l'évolution du procédé à corriger et de proposer des solutions de re-conception comme base pour une re-ingénierie du procédé. Par exemple, une déviation peut devenir une pratique courante plutôt que d'être une exception rare. Dans ce cas, la valeur ajoutée d'un système de workflows devient incertaine et une adaptation est exigée. Ainsi une conception pré-spécifiée, qui est une tâche difficile et pouvant être source d'erreurs, reste insuffisante et mal adaptée face à l'évolution des procédés.

Notre objectif est de développer une approche (théorie et outils) qui permet d'une part de spécifier un modèle de workflow transactionnel qui permet de spécifier facilement et efficacement des mécanismes de recouvrement assurant une exécution fiable et d'autre part de proposer des techniques de découverte et de (re)conception de workflow par l'analyse des traces d'exécutions. Nous nous appuyons pour ces travaux sur l'existant en systèmes de gestion de workflows, modèles transactionnels avancés, la découverte de procédé, et plus généralement les travaux dans le domaine de la fouille de données.

Pour ce faire, nous avons choisi d'enrichir la description du modèle de workflow classique avec des propriétés transactionnelles pour mieux exprimer les mécanismes de recouvrement. Ce modèle sert de base pour les diverses techniques de découverte de workflow développées pour répondre aux besoins d'évolution et de re-ingénierie et assurer des exécutions correctes de workflows. Nous proposons une méthodologie de conception pour la détection des développements observés lors de la phase d'exécution. L'originalité de notre méthodologie est qu'elle se base sur les technique de découverte qui partent des traces d'exécutions de workflows et détectent les lacunes et les déviations entre le modèle de workflow pré-conçu et découvert. L'idée est que, en analysant les traces d'exécutions, il est possible d'agir pour améliorer et optimiser la conception et par la suite la fiabilité de l'exécution des procédés. Cette analyse nous permet de gagner de la visibilité, de comprendre ou prévoir des problèmes, d'identifier et d'optimiser rapidement les solutions.

Chapitre 3

État de l'art

Table des matières

3.1 Introduction

L'étude de la problématique d'ingénierie de procédés métiers pour une modélisation garantissant une **exécution fiable** et **une conception continue** est un sujet de recherche multidisciplinaire. Il est abordé par de nombreuses thématiques de recherche :

- les systèmes de gestion de procédés (en particulier, les workflows) ;
- les mécanismes de contrôle et de correction des échecs d'exécution (en particulier, les modèles transactionnels avancés) ;
- les systèmes de gestion des transactions dans les procédés (modèle, architecture, but, etc.) ;
- l'analyse et la découverte de procédé à partir des traces d'exécutions.

Dans ce chapitre, nous allons confronter ces approches en étudiant leurs capacités d'intégration des dimensions de notre problématique évoquées dans le chapitre précédent. Pour ce faire, nous allons procéder, dans la section 3.2, par une analyse des approches de modélisation de procédés et leur limites en vue d'assurer un modèle de procédé satisfaisant les critères de correction, de

cohérence et des besoins métiers. Nous étudions en particulier les modèles de workflows transactionnels dans la section 3.3. Par la suite, nous étudierons, dans la section 3.4, les méthodologies et les outils de conception et de re-ingénierie de workflows et leurs limites en vue d'assurer une conception continue.

3.2 Technologies et systèmes de support des procédés métiers

Les systèmes de workflows [JB96; Law97; Coa96; AM97; Ell99; vdAvH02a; Fis00] et les modèles transactionnels avancés (MTA) [ELLR90; (Ed92; GMS87; GMGK+91; Mos81; WS92] ont énormément influencé les technologies actuelles des procédés métiers. Elles abordent le problème d'exécution d'un ensemble de tâches encapsulées comme une seule unité de traitement. Pour des raisons historiques, elles abordent ce problème de deux points de vue différents. Dans ce qui suit, nous présentons ces deux approches. Nous montrons en particulier leurs limites en étudiant leurs capacités de répondre à la fois aux besoins métiers de l'utilisateur et aux besoins de correction et de fiabilité d'exécution.

3.2.1 Workflows

3.2.1.1 Procédés métiers et workflows

Le procédé de workflow ou simplement le workflow est un genre spécial de procédés métiers. Souvent, l'utilisation des termes «procédé métier» et «workflow» se confondent, dans le sens qu'ils sont explicitement utilisés comme synonymes [vdAvH02b] ou qu'ils sont présentés côte à côte sans aucun commentaire distinctif [KEP00]. Une autre interprétation populaire est de voir un workflow comme procédé métier administratif, c.à.d, comme procédé métier qui livre des services ou des produits informationnels [vdAB01b]. Le terme «workflow» est également utilisé pour se rapporter exclusivement à la dimension de contrôle d'un procédé métier, c.à.d, les dépendances entre les activités qui doivent être respectées pendant l'exécution d'un procédé métier [DK00; SM01]. Une interprétation finale doit considérer Les procédés métiers comme des workflows qui peuvent être soutenus par des systèmes de gestion de workflows [LD99]. Bien que nous ne soyons pas enthousiastes au sujet de définir des limites conceptuelles des systèmes de gestion de procédés par des caractéristiques de technologies actuelles de systèmes de gestion de workflows, il est intéressant d'explorer la technologie et les caractéristiques essentielles de ces systèmes, comme la plupart des définitions de BPM inclut clairement les systèmes de gestion de workflows [vdAtHW03].

En effet, un procédé est un ensemble d'activités ordonnées selon un ensemble de règles procédurales pour réaliser un objectif précis. Les étapes d'un procédé ne sont pas toujours prévisibles, en particulier quand il gère le suivi des processus durables allant de quelques jours à plusieurs années (par exemple, l'état d'un remboursement de mission, l'état d'avancement d'un apprenant dans une session de Formation Ouverte et à Distance, l'état d'un crédit bancaire, l'état d'une affaire judiciaire, etc.). De ce fait, la définition d'un procédé ne peut pas être complètement impérative (c.à.d. dictant le déroulement des étapes à suivre pour réaliser le procédé -par exemple, par langage de scripts-), mais, elle doit admettre un mode déclaratif (c.à.d. déterminant des règles ou invariants à respecter -par exemple, critères de cohérence, contraintes d'intégrité, pré et post-conditions, jalons et délivrables, etc.-). Par conséquent, la terminologie de workflow, du fait que la définition des workflow intègre le mode déclaratif, est la base pour définir les systèmes de gestion de procédés métiers.

Le modèle de référence définit un workflow comme «une procédure où des documents, des informations, ou des activités sont transmis entre les participants selon un ensemble de règles définies, pour atteindre ou contribuer à un but commercial/métier global» [WFM95]. Un (modèle de) workflow est la représentation du procédé métier dans un format interpretable par la machine. Concrètement, les systèmes de gestion de workflows sont des collecticiels qui offrent des mécanismes de modélisation et d'automatisation de procédés métiers. Ils sont parmi les systèmes les plus élaborés pour définir et exécuter des procédés. Ils permettent, en particulier, de décrire explicitement les méthodes de travail réalisant un procédé, de les expérimenter et de mesurer leurs qualités. Un workflow est un paradigme pour la modélisation et le support de procédés dans des organismes complexes. Souvent les procédés de workflow ont un caractère métier, mais des concepts de workflow ont été également utilisés pour d'autres types de procédés, par exemple, les procédés scientifiques ou les procédés de production de logiciel.

Si un procédé permet de décrire d'une manière informelle les méthodes de travail d'un groupe de personnes et les règles qui les régissent, un workflow permet de formaliser, de structurer, d'automatiser (dans la mesure du possible) et d'exécuter ces méthodes de travail. Des systèmes de gestion de workflows tels que Staffware, IBM MQSeries, COSA, etc. offrent une modélisation générique et des capacités d'exécution pour des procédés structurés. En plus des systèmes de gestion de workflows purs, beaucoup d'autres systèmes ont adopté la technologie de workflows. Considérons par exemple les systèmes de la planification de ressources d'entreprise (ERP) tels que SAP, PeopleSoft, Baan et Oracle, le logiciel de Customer Relationship Management (CRM) , les systèmes de Supply Chain Management (SCM) et les applications Business to Business (B2B) , etc. qui incluent la technologie de workflow. Ces systèmes de gestion de workflows ont démontré un fort potentiel de modélisation et d'automatisation de leurs procédés. La définition d'un procédé et de son modèle d'exécution est représentée par un modèle de workflow qui a été essentiellement développé pour une configuration sur une base indiquant l'ordre dans lequel les activités doivent être exécutées.

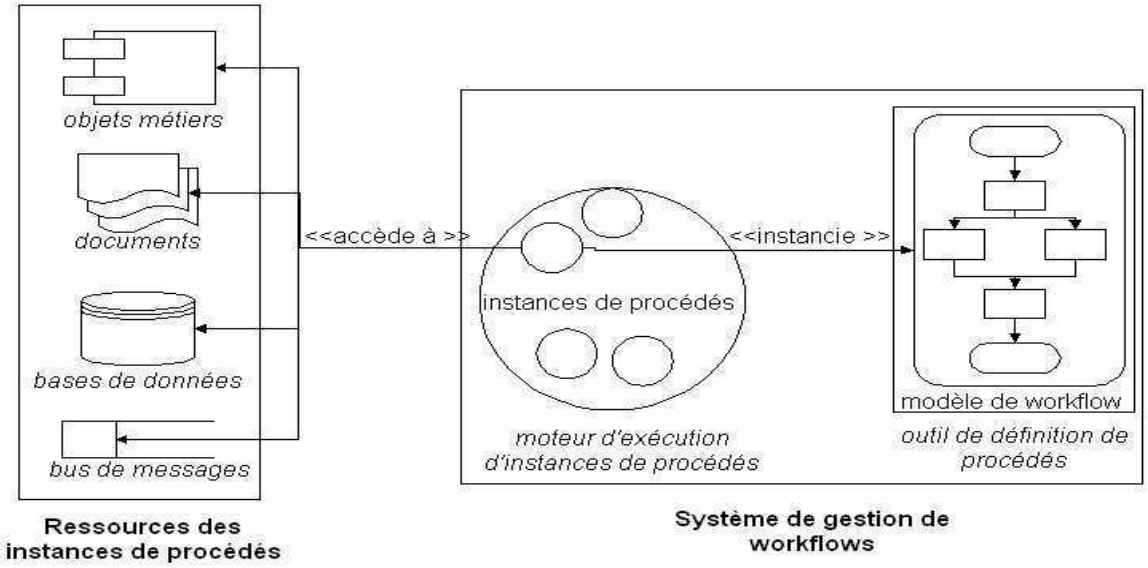
Pour que ce procédé puisse être exécuté, il doit être réalisé par des instances de procédés dont les cycles de vie évoluent au sein d'un moteur d'exécution. Nous considérons, par conséquent, que les outils de définition des procédés et les moteurs d'exécution de leurs instances sont pris en charge par un système de gestion de workflows. Une instance de procédé peut être vue comme une suite d'activités planifiées. Ces activités, exécutées ou interprétées, évoluent sous la tutelle d'un moteur d'exécution à l'intérieur du système de gestion de workflows. La description complète d'une activité ou de son comportement n'est cependant pas du ressort du système de gestion de workflows. En effet, cette exécution peut être décrite d'une manière impérative (par exemple, par une suite d'instructions à réaliser), comme elle peut être décrite d'une manière déclarative par des pré et post-conditions. Comme le montre la figure 3.1, une instance de procédé peut accéder à travers ses activités à des ressources externes à l'outil de gestion de workflows (par exemple, objets métiers, documents, bases de données, bus de messages, etc.) [Bai03; GBBG04]. La description et la maîtrise du comportement de ces activités restent sous la responsabilité du programmeur.

3.2.1.2 Structure d'un workflow

Un modèle de workflow est un graphe dans lequel les nœuds représentent les étapes d'exécution et les arcs représentent le flot de contrôle et le flot de données entre ces étapes. Le meta modèle proposé par la W*f*MC[10] distingue les composants décrits ci-dessous[WFM95] :

[10] **W**ork**f**low **M**anagement **C**oalition : La coalition de gestion de workflows W*f*MC est un groupe de compagnies qui se sont jointes ensemble pour définir des normes pour permettre l'interoperabilité et l'interconnexion de

Figure 3.1 Environnement de définition et d'exécution d'un procédé



- Schéma du workflow : un Schéma du workflow est la représentation d'un procédé dans une forme qui permet sa manipulation automatique (modélisation, exécution) par un système de gestion de workflows. Il est constitué d'un réseau d'activités et des dépendances entre elles, des critères pour spécifier le démarrage et la terminaison d'un procédé et des informations sur les activités individuelles (participants, applications, données informatiques associées, etc.) [WFM95]. Les activités et les procédés ont des données en entrée et en sortie. Les données d'entrée et de sortie sont représentées comme des ensembles d'éléments de données, appelés conteneurs.
- Activité : une activité est une étape dans un workflow. Chaque activité a un nom, un type, des pré et des post conditions et des contraintes d'ordonnancement. Elles peuvent être des activités programme ou des activités procédé. Chaque activité a un conteneur des données en entrée et un conteneur des données résultats.
- Flot de contrôle : un modèle de procédé décrit chaque unité de travail, mais aussi la séquence adéquate d'unités de travail pour atteindre un certain objectif. Un flot de contrôle définit l'ordre dans lequel les activités sont exécutées en spécifiant des connecteurs de contrôle entre les activités.
- Données d'un workflow : À chaque workflow correspond un ensemble de données qui décrivent toutes les informations nécessaires pour son exécution. Ces données incluent (i) des informations requises en entrée des activités, (ii) des informations requises pour l'évaluation des conditions et (iii) des données qui doivent être échangées entre les activités. Ainsi, nous distinguons :
  - le conteneur d'entrée : ensemble de variables et structures typées qui sont utilisées comme entrée à l'application invoquée.
  - le conteneur de sortie : ensemble de variables et structures typées dans lesquelles les résultats de l'application invoquée sont stockés.
- Flot de données : un flot de données est spécifié via des connecteurs de données entre les

différents produits de gestion de workflows

activités mettant en œuvre une série de correspondance entre des conteneurs de données en sortie et des conteneurs des données en entrée permettant l'échange d'information entre les activités. La définition des données du procédé comprend d'une part la spécification des conteneurs d'entrée et de sortie, des activités et des conditions, et d'autre part la spécification du flot de donnée entre les activités.
- Conditions : les conditions spécifient quand certains événements se produisent. Il y a trois types de conditions. `Les conditions de transitions` sont associées aux connecteurs de contrôle et spécifient quand un connecteur est évalué à vrai ou à faux. `Les conditions de démarrage` spécifient quand une activité va être démarrée : par exemple, quand tous ses connecteurs de contrôle entrants seront évalués à vrai, ou quand un d'eux sera évalué à vrai. `Les conditions de sortie` spécifient quand une activité est considérée terminée.

3.2.1.3 Synthèse

L'approche «workflow» donne aux concepteurs des véritables capacités de modélisation lors de la conception pour définir un schéma de workflow spécifiant le procédé métier et facilite, par la suite, l'implantation, la gestion, la distribution et l'exécution de ces procédés métiers. Par ailleurs, l'approche «workflow» émanant des procédés administratifs a particulièrement considéré que la coordination d'activités est fortement liées aux besoins métiers de l'entreprise. En effet, la W*f*MC décrit le système de gestion de workflows comme «un système qui définit, gère et exécute les workflows via l'exécution d'un logiciel dont l'ordre d'exécution est guidé par la représentation en machine de la logique métier du procédé» [WFM95]. Ainsi, les workflows intègrent facilement les besoins métiers par des outils qui permettent de décrire efficacement la méthode de travail.

Cependant, tout en soulignant l'importance des outils de gestion de workflows, nous relevons leurs limites face aux besoins de fiabilité et de conception continue. En effet, les systèmes de gestion de workflows souffrent d'**une phase conceptuelle rigide** face aux besoins d'évolution des utilisateurs et face à la dynamique évolutive des procédés et ont du mal à faire **une description exhaustive** prenant en compte tout les cas et scénarios d'exécution possibles. Généralement, les workflows ont des architectures de spécifications assez rigide et ont des difficultés d'adaptablité face aux changements et aux besoins d'évolution [AAAM97].

De plus, les modèles de workflow (et par conséquence l'approche «workflow») ignorent généralement la dimension «correction». En effet, ils se contentent principalement de modéliser le flot de contrôle et le flot de données et souffrent souvent d'**un déficit de fiabilité** lors de la phase d'exécution. En effet, les outils de modélisation de schémas de workflows actuels ne considèrent pas ou très peu les mécanismes de recouvrement nécessaires pour assurer une exécution fiable et robuste face aux échecs d'exécution.

3.2.2 Modèles Transactionnels Avancés (MTA)

Basés sur le modèle transactionnel ACID, les modèles transactionnels avancés (MTA) peuvent apporter une réponse au besoin de fiabilité qui fait défaut aux workflows. En effet, les MTA ont été proposés comme solution pour dépasser les limites du modèle transactionnel ACID afin de supporter des applications plus complexes que de simples interactions avec une base de données. Ils ont pour but d'assurer des exécutions correctes d'un ensemble de tâches encapsulées comme une seule unité de traitement appelée transaction et ainsi répondre aux besoins de correction et de fiabilité.

3.2.2.1 Genèse des MTA

L'approche transactionnelle a émergé initialement dans le contexte des bases de données. Le modèle des transactions ACID permet d'encapsuler une séquence d'opérations de lecture ou d'écriture sur une base de données comme une unité atomique de traitement, appelée transaction, et qui vérifie les propriétés suivantes :

- Atomicité : l'exécution d'une transaction est atomique ; soit toutes les opérations de la transaction sont exécutées soit aucune n'est exécutée,
- Cohérence : une transaction prise individuellement fait passer la base de données d'un état cohérent vers un autre état cohérent,
- Isolation : les effets d'une transaction en cours d'exécution sont invisibles aux transactions concurrentes,
- Durabilité : les effets d'une transaction validée sont permanents et ne peuvent plus être remis en cause, ni par une défaillance, ni par une autre transaction.

L'intérêt de l'approche transactionnelle est qu'elle assure (i) une exécution correcte (en terme de fiabilité) d'une transaction prise individuellement et (ii) des exécutions correctes (en terme de cohérence de données) de plusieurs transactions concurrentes[11]. L'approche transactionnelle a été étendue et adoptée par des applications plus avancées voulant bénéficier de cette fiabilité et de cette correction. Ces applications varient de simples interactions avec plusieurs bases de données à l'automatisation de procédés métiers. Cependant, le modèle ACID s'est avéré limité pour supporter des telles applications qui (i) nécessitent des structures de contrôle plus complexes qu'une simple séquence d'opérations, (ii) ont des exécutions de longues durées et (iii) où la sémantique de l'annulation n'est pas aussi évidente que pour les opérations de bases de données classiques.

Les modèles transactionnels avancés (MTA) ont été alors proposés pour répondre à ces nouveaux besoins. Dés lors, une transaction désigne plus généralement toute unité de travail encapsulant un ensemble d'opérations. Les MTA permettent de grouper des opérations dans une structure hiérarchique et de relâcher, dans la plupart des cas, certaines des propriétés ACID (généralement l'isolation et l'atomicité). Ces modèles peuvent être classés selon plusieurs caractéristiques, à savoir la structure de la transaction, la concurrence intra transaction, les dépendances d'exécution, les besoins d'isolation et de semi atomicité [WS97a].

Dans la suite, nous présentons quelques modèles transactionnels qui nous semblent être les plus appropriés pour supporter des applications à flot d'activités.

3.2.2.2 Modèle des transactions emboîtées

Une transaction emboîtée [Mos81] est un ensemble de sous-transactions qui peuvent récursivement contenir d'autres sous-transactions, formant ainsi un arbre de transactions emboîtées. Une transaction fille ne peut démarrer qu'après que sa transaction mère ait démarré. Une transaction mère ne peut se terminer que lorsque toutes ses transactions filles sont terminées. La validation d'une transaction est conditionnée par la validation de sa transaction mère. L'abandon d'une transaction mère entraîne l'abandon de toutes ses transactions filles. Cependant, une transaction mère peut survivre à l'échec d'une de ses transactions filles en exécutant, par exemple une sous-transaction alternative. Une transaction est isolée par rapport à toutes transactions n'appartenant pas à sa descendance.

Les transactions emboîtées ouvertes [WS92] relâchent l'isolation en rendant visibles les résultats des sous-transactions validées aux autres transactions emboîtées s'exécutant en parallèle.

[11] Notre problématique concerne plutôt le premier point.

Ainsi une sous-transaction peut valider et libérer ses ressources avant que sa transaction mère se termine correctement. En cas d'abandon d'une transaction, le travail effectué par ses sous-transactions validées peut être sémantiquement compensé en exécutant des sous-transactions de compensation. Une transaction de compensation t^- annule sémantiquement le travail d'une transaction t. Ainsi l'état d'une base de données avant et après l'exécution de la séquence tt^- est un état éventuellemnt différent de l'état initial avant l'exécution, mais considéré comme équivalent et cohérent.

L'accès aux résultats de transactions qui sont ultérieurement compensées peut entraîner des incohérences. Pour éviter ce problème, seules les transactions qui commutent avec celles qui valident peuvent accéder à leurs résultats. Deux transactions s et t commutent si la séquence d'exécution $s\ t$ est équivalente (a le même effet au niveau des résultats et de l'état de la base de données) à la séquence d'exécution $t\ s$. Le problème est ainsi résolu puisque la séquence d'exécution $t\ s\ t^-$ sera équivalente à $t\ t^-\ s$ et par suite équivalente à s.

La contribution majeure des transactions emboîtées est l'extension de la structure plate des transactions ACID vers une structure hiérarchique. Les principaux avantages des transactions emboîtées par rapport aux transactions ACID sont :

- Un niveau de modularité plus élevé : chaque transaction peut être décomposée en une structure hiérarchique de sous-transactions.
- La gestion d'erreurs à un niveau de granularité plus fin : les actions de recouvrement sont prises au niveau de la sous-transaction échouée.
- Le parallélisme intra transaction : les sous-transactions non-conflictuelles peuvent s'exécuter en concurrence.

Bien que le modèle des transactions emboîtées assure plus de flexibilité par rapport aux transactions ACID, il reste limité à des applications base de données et loin de supporter des applications telles que les procédés métiers. Ceci est dû à la structure de contrôle primitive de ce modèle par comparaison à celle des modèles de workflows.

3.2.2.3 Modèle des Sagas

Le modèle des Sagas [GMS87] a été proposé comme une solution pour les transactions de longue durée. L'idée de base est de permettre à une transaction de libérer certaines ressources qu'elle a acquises avant de valider. Une transaction de longue durée ou Saga est une séquence de sous-transactions, T_1, ..., T_n, qui peuvent s'intercaler avec d'autres transactions. Chaque sous-transaction T_i est une transaction ACID qui préserve la cohérence de la base de données. Les exécutions partielles d'une saga sont inacceptables. Si une saga abandonne (suite à l'échec de l'une de ses sous-transactions) alors le travail réalisé jusqu'alors (par les précédentes sous-transactions) doit être compensé. Ainsi, chaque sous-transaction T_i possède une transaction de compensation C_i qui annule sémantiquement son travail. Plus formellement, le système assure que : soit la séquence T_1, ..., T_n est exécutée en cas de non-problème, soit T_1, T_2, ..., T_i, C_i, ..., C_2, C_1 en cas d'échec de la sous-transaction T_i. Dans sa première version, un seul niveau d'emboîtement est autorisé. Ce modèle initial a été ensuite étendu [GMGK+91].

Le modèle des Sagas, comme les transactions emboîtées, adopte une structure hiérarchique pour définir une transaction structurée. Il repose sur la compensation pour relâcher la propriété d'isolation. Ce qui permet de réduire l'isolation au niveau des sous-transactions, d'éviter le blocage de ressources inutilement pour de longues durées et d'augmenter ainsi la concurrence inter transactions.

Cependant, le modèle des sagas présente certaines lacunes qui limitent l'étendue de son application. En effet, il présente comme les transactions emboîtées, une structure de contrôle

limitée par rapport aux besoins des procédés métiers. En plus il suppose certaines contraintes qui ne sont pas forcement respectées dans beaucoup d'applications. Par exemple, il faut pouvoir découper à l'avance l'application en sous-transactions, où chacune possède une transaction de compensation. En plus la propriété d'atomicité qu'il préserve n'est pas appropriée au contexte des procédés métiers notamment de longue durée.

3.2.2.4 Modèle des transactions flexibles

Le modèle des transaction flexibles [ELLR90; ZNBB94; ARS92; SABS02] a subi plusieurs évolutions depuis sa proposition. Initialement, les transactions flexibles [ELLR90] ont été proposées comme un modèle transactionnel convenable dans un environnement multi bases de données. Une transaction flexible est un ensemble d'activités, chacune ayant un ensemble de sous-transactions fonctionnellement équivalentes. Un ensemble de dépendances d'exécution sur les sous-transactions, incluant des dépendances d'échec, de succès, ou des dépendances liées à des événements externes, peut être spécifié. Pour relâcher l'isolation, les transactions flexibles utilisent la compensation. L'atomicité globale est relâchée en permettant la spécification d'états de terminaison acceptés dans lesquels certaines transactions peuvent avoir avortées.

Le modèle des transactions flexibles a été ensuite étendu et adopté pour supporter des applications plus avancées [ZNBB94; ARS92]. [SABS02] adopte les sémantiques transactionnelles d'activité **rejouable**, **compensable** et **pivot** [MRKS92]. Une activité t est dite **rejouable** si elle se termine toujours avec succès après un nombre fini d'activations. t est dite **compensable** si son travail peut être sémantiquement annulé. Une activité est dite **pivot** si une fois qu'elle se termine avec succès, ses effets ne peuvent pas être compensés [SABS02].

Le modèle des transactions flexibles est un modèle hybride à mi-chemin entre l'approche transactionnelle et les systèmes de gestion de procédés. L'originalité de ce modèle est l'utilisation de propriétés transactionnelles en plus de la compensation permettant d'atteindre plus de flexibilité tout en préservant un certain degré de correction.

Le degré de flexibilité élevé, par comparaison aux modèles des sagas et des transactions emboîtées, a élargi l'étendue des transactions flexibles au-delà des applications centrées bases de données. Cependant, malgré ce succès relatif ce modèle reste limité à quelques applications spécifiques et inadéquat pour les modèles à flot d'activités à large échelle. Comme les deux modèles précédents, ceci est dû à sa structure de contrôle qui reste primitive par rapport à celle des workflows et à l'ensemble des contraintes (par exemple imposer que chaque activité spécifie indépendamment du reste du procédé sa compensation) qu'il définissent et qui doivent être respectées par les concepteurs d'applications.

3.2.2.5 Synthèse

Par l'étude précédente de quelques MTA, nous pouvons conclure que les MTA, qui sont des modèles généralement basés sur le modèle transactionnel ACID, ont pour but d'assurer des exécutions correctes d'un ensemble de tâches encapsulées comme une seule unité de traitement appelée transaction et ainsi répondre aux besoins de correction et de fiabilité. Le modèle transactionnel traditionnel a évolué au cours du temps pour incorporer des structures de transactions plus complexes et relâcher les propriétés d'atomicité et d'isolation. Les MTA, résultats de cette évolution, ont permis de dépasser les limites du modèle ACID pour satisfaire les besoins de nouvelles applications avancées. Bien que ces modèles aient été proposés dans des contextes plus ou moins similaires, ils partagent certaines caractéristiques :

– une structure plus complexe qu'une simple séquence d'opérations,

- un degré de concurrence et de parallélisme intra-transaction plus élevé,
- un relâchement des propriétés d'atomicité et d'isolation,
- une adoption de nouvelles sémantiques transactionnelles comme la compensation et l'alternative.

Cependant, malgré le degré de flexibilité qu'ils apportent, ces modèles restent limités dans le cadre des modèles à flot de activités [RS95b; WS97a; GC02; SABS02] à large échelle. Leurs limites découlent principalement :

- de leurs structures de contrôle limitées : bien que les MTA aient étendu la structure plate du modèle ACID vers des structures plus complexes, elles restent primitives par comparaison à celles des procédés métiers qui nécessitent des constructeurs de branchement et de synchronisation plus évolués. Par exemple, aucun des modèles présentés ci-dessus n'est capable de modéliser le prêt bancaire, aussi simple soit-il.
- des contraintes qu'ils exigent : les MTA imposent des contraintes, au niveau de la structure et des sémantiques transactionnelles adoptées, que les concepteurs doivent respecter lors de la spécification de leurs applications. De plus, en adoptant un modèle donné, les concepteurs doivent adhérer au critère de correction implicitement défini par l'ensemble des règles prédéfinies. Ceci rend les MTA plus appropriés pour des approches descendante[12] que pour des approches ascendante[13]. Par approche descendante nous entendons une approche qui part d'un MTA et essaye de modéliser l'application selon ce modèle en respectant ses règles et ses contraintes. Par approche ascendante, nous entendons une approche qui part d'une application existante et essaye d'assurer sa fiabilité.

En conclusion, nous pouvons dire que les MTA ont étendu le modèle ACID pour assurer plus de flexibilité. Cependant, ils restent handicapés par une structure de contrôle qui reste primitive par rapport à celle des workflows et n'arrivent pas à intégrer d'une manière efficace les besoins d'une sémantique métiers des procédés.

3.2.3 Synthèse

Pour des raisons historiques, les modèles transactionnels avancés (MTA) et les systèmes de workflows ont abordé la problématique de procédés métiers de deux points de vue différents. Basés sur le modèle transactionnel ACID, les MTA ont pour but d'assurer des exécutions correctes d'un ensemble d'activités encapsulées comme une seule unité de traitement appelée transaction, en offrant par rapport aux transactions classiques un degré de flexibilité accru. Issus des procédés administratifs, les workflows offrent des mécanismes de spécification répondant au mieux au besoins métiers des procédés permettant de modéliser des procédés métiers complexes. Bien que ces deux approches apportent, en de nombreux points, réponses à certains nombre de nos besoins, nous avons montré en particulier, à ce niveau, comment séparément ces deux approches n'ont pas pu intégrer les dimensions «procédés métiers» et «correction».

Principalement, les systèmes de workflows ont toujours été sujets à de nombreuses critiques à cause de l'absence de mécanismes de correction et de fiabilité [AAAM97]. Bien que, les modèles de workflow, par comparaison aux MTA, aient une sémantique plus riche et abordent des problèmes qui ne sont pas pris en compte par les modèles transactionnels avancés, ils ont ignoré les problèmes liés à la fiabilité des exécutions en cas d'échecs. En fait, dans le contexte des procédés métiers, les échecs des activités ont été résolus de façon ad-hoc, au cas par cas et éventuellement via des interventions humaines. C'est pourquoi, les systèmes de workflows ont été toujours critiqués à cause de ce manque de fiabilité.

[12] Le terme anglophone est top-down
[13] Le terme anglophone est bottom-up

Une réflexion intuitive nous suggère d'intégrer ou de combiner les modèles transactionnels et les workflows pour créer une approche combinée de modélisation des procédés métiers garantissant à la fois tous ces besoins. Ainsi un nouveau concept a émergé qui est celui du workflow transactionnel. Les workflows transactionnels incorporent des sémantiques transactionnelles pour intégrer la dimension «correction» afin d'assurer un certain degré de fiabilité. La section suivante présente une étude détaillé de ce modèle.

3.3 Workflows transactionnels

3.3.1 Introduction

Le terme de workflow transactionnel [SR93] a été introduit pour reconnaître la pertinence des transactions dans le contexte des workflows pour assurer un certain degré de correction et de fiabilité. Les workflows transactionnels concernent l'exécution coordonnée de plusieurs activités et suggèrent l'utilisation sélective de modèles transactionnels pour assurer des mécanismes de correction et de fiabilité. Depuis, l'approche workflow transactionnel à été sujet de nombreux travaux de recherche [RS95b; KS95; RS95a; BZB97; WS97b; SKM+96; DDGJ01; SABS02; HB04; PG06; VG03]. Les workflows transactionnels fournissent les mêmes fonctionnalités que les workflows en terme d'automatisation des procédés métiers. En plus, ils visent à assurer des exécutions fiables et correctes en présence de la concurrence et des échecs. Ceci n'implique pas que les workflows sont similaires ou équivalents aux transactions des bases de données, ou qu'ils supportent toutes les propriétés ACID. Néanmoins, des tels workflows partagent les objectifs de quelques MTA dans le sens où ils veillent à imposer des contraintes transactionnelles à un ensemble d'activités. Par comparaison aux modèles transactionnels avancés, les workflows transactionnels s'intéressent aux problèmes de cohérence d'un point de vue métier plutôt que d'un point de vue base de données. La portée des workflows transactionnels s'étend au-delà de celle des transactions des bases de données et des MTA.

La combinaison de ces deux concepts reste toutefois très diversifiée et confuse. Dans la suite, nous reprenons des travaux de Greffen et autres [PG06] pour définir un cadre de classification et donner une taxonomie des modèles de workflows transactionnels existants. Cette classification distingue entre un point de vue conceptuel, qui est centré autour des aspects de langages, et un point de vue système porté sur des aspects d'architecture. Cette analyse fournit une base claire pour comparer les différentes classes, et systèmes existants de workflows transactionnels et les positionner par rapport aux besoins de notre problématique.

3.3.2 Point de vue conceptuel

D'un point de vue conceptuel, deux approches, ayant une architecture conceptuelle différente, ont été présentées pour spécifier le modèle du workflow transactionnel :

- une **approche aux modèles séparés** (plus de détails dans la section 3.3.2.1) qui traite séparément le modèle de workflow et le modèle transactionnel. Ces modèles séparés sont combinés pour obtenir le modèle du workflow transactionnel.
  Cette classe est raffinée en trois sous classes selon le type de la relation d'abstraction entre le modèle de workflow et le modèle transactionnel dans le workflow transactionnel :
  - **Workflows par-dessus les Transactions (WF/TR)** : le modèle transactionnel est utilisé pour fournir la sémantique transactionnelle nécessaire aux modèles de workflows.
  - **Transactions par-dessus les Workflows (TR/WF) :** le modèle de workflow est utilisé pour fournir la structure de procédé aux modèles transactionnel.

  - **Transactions et Workflows en pair à pair (TR+WF) :** le modèle de workflows et le modèle transactionnel existent au même niveau.
- une **approche aux modèles intégrés** (plus de détails dans la section 3.3.2.2) qui voit le modèle de workflow et le modèle transactionnel en tant que deux aspects complémentaires et intégrés. Dans cette architecture conceptuelle un modèle unique est utilisé. Il n'y a pas de relations entre le modèle de workflow et le modèle transactionnel. Cependant il y a trois variantes relatives au modèle dominant utilisé :
  - **Modèle hybride de workflow transactionnel (TRWF) :** un modèle hybride simple contient des concepts du modèle de workflow et du modèle transactionnel.
  - **Transactions dans les workflows (WF) :** Un modèle simple de workflow est utilisé, dans lequel des aspects du modèle transactionnel correspondent à des primitifs du modèle de workflow.
  - **Workflows dans les transactions (TR) :** Un modèle simple du modèle transactionnel est utilisé, dans lequel des aspects du modèle de workflow correspondent aux primitives du modèle transactionnel.

Ces classes conceptuelles peuvent correspondre directement à des spécifications de langages relatives aux modèles de workflows et aux modèles transactionnels. Étant donné les deux classes conceptuelles principales, nous pouvons avoir deux situations :

- Il y a deux langages différents : un langage pour spécifier des aspects de workflows, le langage de définition de workflow (WFDL), et un langage pour spécifier des aspects des transactions, le langage de définition de transaction (TRDL).
- Il y a un seul langage intégré pour la spécification des aspects de workflow et de transaction, le langage de définition de workflow transactionnel (TRWFDL).

Si le TRDL est un raffinement du WFDL, le niveau de WFDL contient des attributs de workflow et les états intermédiaires au niveau de TRDL sont liés aux états de transaction. Si le WFDL est un raffinement du TRDL, le niveau de TRDL contient des attributs transactionnels et les états intermédiaires au niveau de WFDL sont liés aux états du flot de contrôle. Dans l'approche intégrée, tous les aspects sont fusionnés dans un seul langage.

3.3.2.1 Approche conceptuelle aux modèles séparés

Dans cette section, nous présentons l'approche conceptuelle **aux modèles séparés** (c.à.d modèle de workflow et modèle transactionnel) pour la spécification de workflows transactionnels. Les raisons principales pour l'usage de deux langages séparés sont une flexibilité de couplage accrue entre les modèles et une séparation des intérêts entre les aspects du flot de contrôle et du comportement transactionnel pour des applications complexes. Dans ce qui suit, nous détaillons les trois sous classes de cette approche. La première sous classe considère les workflows transactionnels comme des workflows enrichis par un comportement et une sémantique transactionnelle. La deuxième sous classe part des modèles transactionnels en essayant de les étendre. La troisième est une combinaison des deux modèles séparés.

Workflows par-dessus les Transactions (WF/TR)

Cette classe (**WF/TR**) utilise un modèle de workflow qui est étendu en utilisant des sémantiques transactionnelles définies par un modèle transactionnel pour assurer des besoins de fiabilité et de correction. Dans cette classe, l'aspect de flot de contrôle mène aux spécifications des workflows transactionnels. Par conséquent, le TRDL est un raffinement du WFDL. Des primitives du WFDL sont mappées sur des primitives du TRDL. Cette approche est adoptée dans

langage de description de workflow	langage de description transactionnel
BEGIN TASK BUSINESS PROCESS STB USES FORM form1 END TASK	BEGIN BUSINESS PROCESS STB READ client.nom READ client.nom USE form1 WRITE banque.client WRITE banque.pret IF status_ok THEN terminate STB ELSE activate STB END TRANSACTION

TAB. 3.1 – Les workflows transactionnels comme extension des workflows

la plupart des systèmes de gestion de workflows commerciaux qui supportent un comportement transactionnel.

Le degré auquel chacun de ces modèles de workflows incorpore les caractéristiques transactionnelles varie et dépend largement des besoins (comme la flexibilité, l'atomicité et l'isolation des exécutions des activités individuelles et des multiples instances de workflows, etc.) du procédé métier qu'il essaye de modéliser.

Dans le tableau 3.1 nous donnons un exemple simple de cette classe inspiré de notre exemple de motivation de prêt bancaire. Du côté gauche, nous voyons une spécification de l'activité STB du workflow en WFDL «simplifié». Les deuxième et troisième lignes de ces spécifications sont étendues à un niveau d'abstraction plus bas selon les spécifications transactionnelles décrites dans la colonne droite du tableau. Une fois exécutée, les spécifications de TRDL induisent des états intermédiaires selon les spécifications de WFDL. Notons que certains langages permettent à des activités multiples d'être groupées dans une simple transaction métier (par exemple [GVBA97]).

L'architecture de référence de Mercurius [GdV98] est un exemple clair d'une architecture dans la classe de **WF/TR** ; le modèle de workflow est placé au dessus du modèle transactionnel. Aussi bien que le système de gestion de workflows METEOR [SKM+96] dans lequel les activités sont modélisées en fournissant leurs états visibles et les événements correspondants aux transitions d'états [RS95b; KS95]. Les dépendances inter-activités peuvent être classées comme des dépendances qui peuvent être activées, rejetées ou retardées. Ce modèle supporte des activités ayant des sémantiques transactionnelles et non transactionnelles.

Pour relâcher l'atomicité, METEOR utilise la notion d'états de terminaison acceptés (ETA) comme critère de correction. Tout état n'appartenant pas à ETA est un état de terminaison non accepté, dans lequel la semi atomicité est non respectée. On distingue les états de terminaison acceptés `de validation` et les états de terminaison acceptés `d'abandon`. Un état de terminaison `de validation` est un état dans lequel le workflow a atteint ses objectifs. Au contraire, un état de terminaison accepté `d'abandon` est un état de terminaison valide dans lequel le workflow a échoué et n'a pas atteint ses objectifs. Si un état de terminaison accepté `d'abandon` est atteint, tous les effets indésirables de l'exécution partielle du workflow doivent être annulés conformément aux besoins définis.

Le système METEOR définit la logique d'exécution en spécifiant des dépendances entre les activités (basées sur leurs états). Ce qui permet de définir aussi bien un flot de contrôle que des mécanismes de recouvrement. De plus le système METEOR définit un critère de correction des exécutions. Cependant, le grand problème du système METEOR est que le mécanisme développé

langage de description transactionnel	langage de description de workflow
BEGIN TRANSACTION EXECUTE ATOMIC IMPLEMENTATION wf1 END TRANSACTION	BEGIN WORKFLOW wf1 TASK TASK STP VSC VD ... AND-SPLIT STP VSC VD SEQUENCE ER MJR AND-SPLIT RD VD LD .... END WORKFLOW

TAB. 3.2 – Les workflows transactionnels comme extensions des MTA

pour assurer le critère de correction n'est pas approprié au contexte des procédés métiers. En effet, pour assurer la fiabilité, le système METEOR vérifie à la fin de l'exécution si l'application s'est terminée dans un état acceptable ou non. Si oui ses effets sont validés, sinon ils sont annulés. Cette façon de procéder n'est pas appropriée au contexte des procédés métiers à cause de leurs exécutions de longue durées. En plus, il n'est pas toujours possible d'annuler le travail effectué. C'est pourquoi le système METEOR reste limité aux applications centrées bases de données.

Transactions par-dessus les Workflows (TR/WF)

Dans cette classe (**TR/WF**), le modèle transactionnel est enrichi pour incorporer des concepts du modèle de workflow. Ainsi, le comportement transactionnel est le principal aspect dans les spécifications des workflows transactionnels. La sémantique transactionnelle de haut niveau du modèle transactionnel est spécifiée avec le modèle de workflow pour l'élaboration de la structure fondamentale du workflow transactionnel. Par conséquent, le WFDL n'est qu'un raffinement du TRDL. L'approche de **TR/WF** est appliquée, par exemple, dans la gestion de workflows pour des applications de commerce électronique. Ici, la transaction entre deux associés commerciaux est le point de départ et l'élaboration du flot de contrôle n'est qu'une amélioration de la transaction.

Dans le tableau 3.2 nous donnons un exemple simple de cette classe inspiré de notre exemple de motivation de prêt bancaire. Du côté gauche, nous voyons des spécifications d'un TRDL «simplifié» d'une transaction qui exprime des propriétés transactionnelles, en particulier la propriété atomique appliquée à toute la transaction dans ce cas. Dans cette transaction, le flot de contrôle est comme un détail d'implémentation spécifié à un niveau d'abstraction plus bas dans la colonne de droite du tableau en WFDL «simplifié». Les spécifications en WFDL décrivent un procédé non linéaire, qui n'est pas facile à spécifier dans les TRDL traditionnels. Cet exemple nous montre que l'exécution de la transaction exprimée en WFDL présentera des états intermédiaires décrits dans le workflow exprimé en TRDL.

L'approche CrossFlow [VG03] peut être considérée comme un exemple de la classe de **TR/WF**. Dans la perspective de langage CrossFlow, des transactions inter-organisationnelles sont spécifiées dans un contrat électronique qui correspond aux définitions de workflow.

Transactions et Workflows en pair à pair (TR+WF)

Dans cette classe de (**TR+WF**), il y a un équilibre entre le flux de contrôle et le comportement transactionnel. La sémantique transactionnelle de haut niveau est définie au même niveau conceptuel que les procédés de workflows. Par conséquent, le flux de contrôle et les caractéristiques transactionnelles se rapportent au même niveau d'abstraction.

langage de description de workflow	**langage de description transactionnel**
BEGIN WORKFLOW banque	BEGIN TRANSACTION tr1
REFERS TRANSACTION tr1	REFERS WORKFLOW banque
TASK TASK *STP VSC VD* ...	COMP *CR RD*
AND-SPLIT *STP VSC VD*	SAFEPOINT *VSC*
SEQUENCE *ER MJR*	END TRANSACTION
AND-SPLIT *RD VD LD*	....
....	

TAB. 3.3 – Les workflows transactionnels comme extension combiné de modèles transactionnels et de workflows

Dans le tableau 3.3 nous donnons un exemple simple de cette classe inspiré de notre exemple de motivation de prêt bancaire. Du côté gauche, nous définissons le flux de contrôle WFDL «simplifié» qui réfère le flux transactionnel spécifié en TRDL «simplifié» pour définir en particulier le mécanisme de recouvrement en cas d'échec de l'activité RD. D'une manière analogue, le flux transactionnel spécifié en TRDL dans la colonne de droite importe la liste des activités de la colonne de gauche pour spécifier le mécanisme de compensation (comme à l'origine proposé dans le modèle de Saga [GMS87]) et un point cohérent[14] (comme proposé dans le modèle WIDE [GVA01]) qui permet de définir un recouvrement flexible par compensation de l'activité RD. Notons que le flux de contrôle et les spécifications de la fonctionnalité de compensation peuvent être changés indépendamment les uns des autres et de ce fait créer une séparation entre le workflow et les spécifications transactionnelles.

Dans [DDGJ01], on propose une approche qui s'inscrit dans la classe de **TR+WF** dans laquelle l'indépendance entre les spécifications de flot de contrôle d'une part, et les caractéristiques transactionnelles d'autre part sont le point de départ. L'approche est basée sur les sphères transactionnelles. De même, les ATMs (Activities/Transaction Model) [DHL90; DHL91] permettent d'intégrer des transactions ACID et des activités qui ne doivent pas être ACID. Le flot de contrôle et le flot de données peuvent être spécifiés statiquement dans un script ou dynamiquement par des règles événement-condition-action. Le recouvrement est supporté par la spécification de règles de compensation ou de transactions de contingence qui représentent des alternatives.

3.3.2.2 Approche conceptuelle aux modèles intégrés

Dans cette section, nous présentons l'approche conceptuelle **aux modèles intégrés** pour la spécification de workflows transactionnels. Les modèles de workflows et les modèles transactionnels sont intégrés dans un seul modèle. Ceci permet un niveau d'intégration élevé entre les deux aspects, mais il introduit souvent de la complexité et l'inflexibilité.

Modèle hybride de workflow transactionnel (TRWF)

Dans cette classe (**TRWF**), nous trouvons des modèles hybrides de workflows et des transactions. Ces modèles sont représentés dans les langages de spécification hybrides de workflows transactionnels. Ces langages contiennent des primitives typiques relatives aux workflows. Une manière évidente de créer un langage de (**TRWF**) est de fusionner les langages de la classe

[14] Le terme anglophone est «safepoint»

TR+WF. Clairement, les approches de TRWF et de TR+WF sont échangeables dans une certaine mesure.

Un exemple de ce type de classe est le projet Exotica [MAGK95; AAA+96] qui décrit des méthodes et des outils pour implanter des modèles de transactions avancées au dessus de Flowmark[15]. L'idée de base a été de fournir un modèle de workflow étendu qui intégre des concepts de transactions avancées. En particulier, les auteurs ont montré comment les SAGA et les transactions flexibles pourraient être implantées au dessus de Flowmark. L'utilisateur peut définir une activité de compensation pour chaque activité de procédé. Un preprocesseur transforme ces spécifications en FDL (Flowmark Definition Language) en insérant des chemins de compensation après une activité ou groupe d'activités, qui sont exécutées de manière conditionnelle après l'échec d'une activité.

Dans l'approche d'Exotica [AM97], on propose un langage dans la classe de **TRWF**. Ce langage est pré-traité pour être interprétable par une architecture dans la classe de WF. Dans le projet de FlowBack [KMO98a], une approche semblable a été développée. De même, le projet WIDE [GPS99] spécifie un système de gestion de workflows qui combine les aspects des deux langages.

Transactions dans les workflows (WF)

Dans la classe (**WF**), la sémantique transactionnelle est exprimée dans le procédé de workflow. Des patrons de procédés spécifiques sont utilisés dans ce cas pour exprimer le comportement transactionnel d'un procédé de workflow.

Un exemple typique de cela est la spécification des patrons de compensation dans des définitions de workflows pour relaxer les caractéristiques d'atomicité pour un workflow ou les patrons de workflow d'abondon proposés dans [VTKB03]. Par ailleurs, les réseaux de Petri colorés peuvent aussi intégrés des mécanismes transactionnels. En effet, Dehnert code des caractéristiques transactionnelles en utilisant les réseaux de Pétri [Deh01].

Workflows dans les transactions (TR)

Pour cette classe (TR) de la taxonomie, un modèle simple du modèle transactionnel, dans lequel des aspects du modèle de workflow correspondent aux primitives du modèle transactionnel, est utilisé. Le modèle de «contrats» [RS95a] est un exemple de cette classe. Un contrat est une exécution cohérente et tolérante aux pannes d'une séquence d'actions prédéfinies selon un scénario prédéfini (script) qui décrit l'enchaînement des étapes [WR92]. Les étapes constituent les éléments unitaires d'un contrat. Une étape est une transaction ACID qui implémente un traitement particulier (réservation de vol, réservation d'hôtel). Un script décrit la structure (flot de contrôle) du contrat. Un des points clefs du modèle réside dans la notion de recouvrement en avant. En cas d'incident, le gestionnaire de contrat réinstancie les contrats interrompus dans l'état cohérent le plus récent et continue l'exécution du script.

En outre, le modèle des contrats intègre assez bien les dimensions «flexibilité» et «procédés métiers». En effet, comme le modèle de workflow, il permet de spécifier la logique d'exécution d'une application. De plus, le modèle des contrats intègre des sémantiques transactionnelles. Ainsi, il considère chaque étape du contrat comme une transaction ACID. Il permet de regrouper certaines étapes dans une même transaction ACID et/ou de définir des relations entre les étapes. Il permet de définir ainsi des mécanismes de recouvrement en avant.

[15] Le système de workflow qui est le prédécesseur de IBM MQ Series

Par comparaison aux modèles transactionnels, le degré de fiabilité du modèle de «contrats» est limité. En effet, les mécanismes de recouvrement sont définis d'une façon ad-hoc par les concepteurs. Le modèle de «contrats» ne définit pas un critère de correction ni des mécanismes pour vérifier la cohérence et la correction des spécifications des concepteurs.

Le TSME (Transaction Specification and Execution Environement) [GHKM94] est un système programmable qui supporte la spécification des workflows transactionnels et la configuration des gestionnaires de transactions pour implanter les modèles désirés. Il fournit un langage de spécification pour exprimer les différents types de dépendances intra et inter-transactions et les critères de correction que la transaction doit satisfaire. En utilisant une interface de programmation C++, ASSET [BDG+94] permet à un concepteur de transaction de compiler des spécifications de transactions avancées dans un code exécutable. ASSET se base sur des primitives de transactions dérivées du formalisme ACTA [16] [CR94].

3.3.3 Point de vue architectural

Après avoir discuté du point de vue conceptuel dans la section précédente, nous nous tournons, dans cette section, vers le point de vue système. Le point de vue conceptuel explique le «Quoi», alors que le point de vue système explique le «Comment», c.à.d, le support système des modèles de workflows et des modèles transactionnels. Nous basons le point de vue système sur l'aspect architectural, se concentrant sur la structure de haut niveau des systèmes supportant le workflow transactionnel.

Dans la description des architectures, nous plaçons les modules de workflows (MWF) et le module transactionnel (MWF) au dessus d'un module de gestion d'échec (SGE). Dans les classes où le modèle workflow et le modèle transactionnel sont séparés, nous avons une relation architecturale entre ces deux modules. Cette relation peut être basée sur deux patrons architecturaux. Dans une architecture verticale (hiérarchique) d'un patron, nous avons une relation client/serveur entre les deux modules. Dans une architecture horizontale d'un patron, deux modules ont une relation pair à pair (ou la relation serveur/serveur).

3.3.3.1 Systèmes séparés

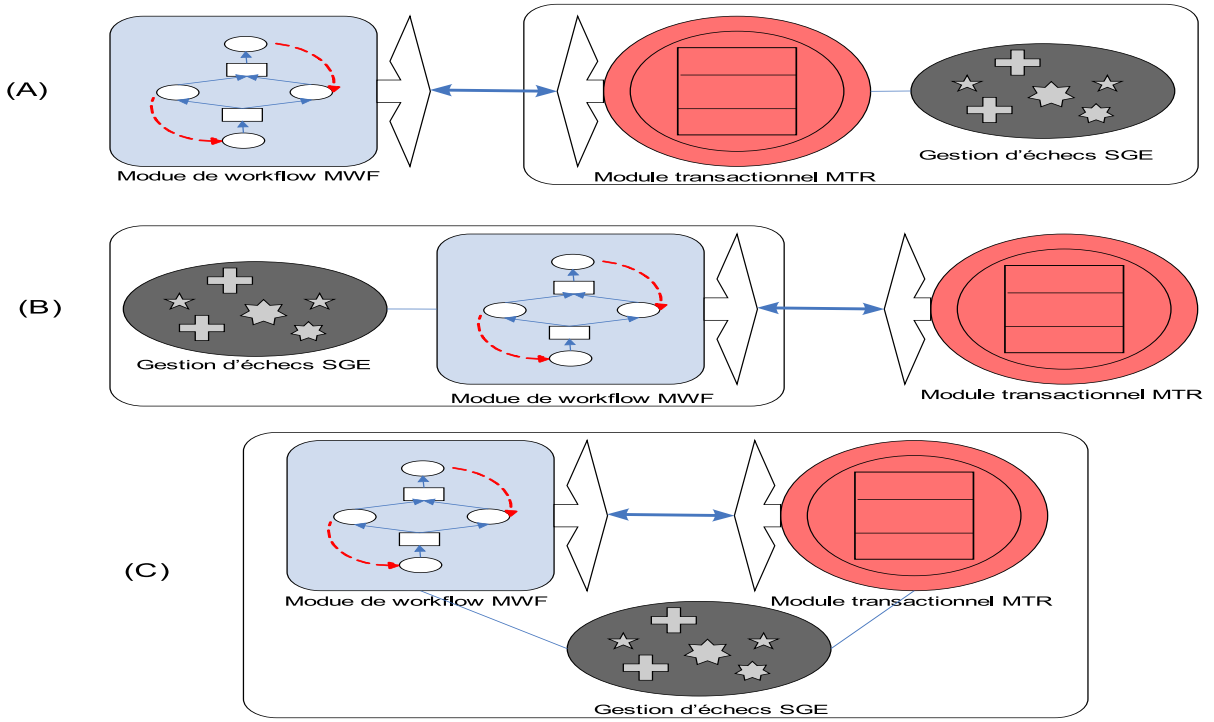
Dans la catégorie des modèles séparés, nous avons un module de workflow **MWF** et un module transactionnel **MTR** séparés. Ces modules peuvent avoir trois relations architecturales (*c.f.* figure 3.2) : **MWF** comme client de **MTR**, **MTR** comme client de **MWF**, et **MWF** et **MTR** comme modules de pair à pair. Ces trois architectures coïncident avec les trois classes **MWF/MTR**, **MTR/MWF** et **MTR+MWF**. En analysant les caractéristiques des trois classes, notre centre d'intérêt se porte sur l'interface de **MWF** et **MTR**. Cette interface est utilisée dans chacune des trois architectures pour synchroniser l'état du flot de contrôle de **MWF** et les états de transactions dans **MTR**.

L'architecture de **MWF/MTR** est décrite dans la figure 3.2.A. Le **MWF** utilise l'interface de **MTR** pour ouvrir un contexte de transaction et pour effectuer des opérations de manipulation de données dans ce contexte. Dans cette classe, les module **MTR** et **SGE** sont souvent intégrés dans un environnement d'application de base de données avec la fonctionnalité intégrée de gestion de transactions. L'architecture de **MWF/MTR** est un standard pour les systèmes commerciaux.

L'architecture de **MTR/MWF** est décrite par la figure 3.2.B. Le **MTR** utilise l'interface de **MWF** pour ouvrir un contexte de workflows et puis invoquer les primitives de flot de contrôle. Nous observons cette classe d'architecture dans des environnements de commerce électronique

[16] ACTA est un métamodèle de transactions qui permet la spécification des modèles de transactions avancées

Figure 3.2 Architecture de workflow transactionnel



où un moteur de transaction de haut niveau invoque des procédés supportés par des technologies de gestion de workflows.

L'architecture de **MTR+MWF** est illustrée par la figure 3.2.C. Dans cette architecture, nous avons un **MTR** comme serveur de transactions et un **MWF** comme serveur de procédés dans une relation pair à pair. L'interface entre **MWF** et **MTR** est uniquement un canal de contrôle : le **WFM** communique les états de procédés au **MTR**, le **MTR** communique des contextes de transactions pour agir par des actions transactionnelles sur les états de procédés. L'architecture de **MTR+MWF** sert généralement comme support au langage **TR+WF**.

3.3.3.2 Systèmes intégrés

Dans la classe de modèles intégrés, nous avons un module transactionnel simple de gestion de workflows. Ceci peut être soit un gestionnaire de workflow transactionnel (**MTRWF**), soit un gestionnaire traditionnel de workflow (**MWF**), ou un gestionnaire de transactions avancées (**MTR**).

Le **MTRWF** offre un support intégré pour la gestion du module transactionnel et du module de workflow. Un état hybride est maintenu dans cette classe qui supporte en particulier les langages des classes conceptuelles de **TRWF** et **TR+WF**.

Dans la classe architecturale **MWF**, les attributs transactionnels de l'état d'un workflow transactionnel sont mis en correspondance avec les attributs de workflows. Le **MWF** peut seulement interpréter des caractéristiques de la classe **WF**. Les caractéristiques des autres classes (typiquement **TRWF**) doivent être traduites au format de **WF**. Cette classe d'architecture est adaptée pour le support des systèmes de gestion de workflows commerciaux.

Dans la classe architecturale **MTR**, les attributs du flot de contrôle de l'état d'un workflow transactionnel sont mis en correspondance avec les attributs du modèle transactionnel. Le **MTR** peut seulement interpréter les caractéristiques de la classe conceptuelle **TR**. Les caractéristiques des autres classes (typiquement **TRWF**) doivent être traduites au format de la classe **TR**. Des procédés embarqués peuvent être supportés par une technologie standard de gestion de transactions.

3.3.4 Synthèse

Dans cette section, nous avons présenté les différentes classes conceptuelles et architecturales des workflows transactionnels. Cette classification sert de base pour analyser les approches existantes et pour choisir le modèle de workflow transactionnel qui répond au mieux aux besoins de :

- flexibilité : laisser la liberté aux concepteurs de spécifier leurs besoins tant au niveau de la structure de contrôle par le choix du modèle de workflow de l'application qu'au niveau du degré de correction exigé par le choix du modèle transactionnel,
- correction : pouvoir fournir des structures transactionnelles assurant des exécutions correctes,
- besoins des procédés métiers : il faut que les techniques développées et les sémantiques utilisées soient appropriées aux caractéristiques des procédés métiers, notamment la longue durée des exécutions et les activités non annulables,
- expressivité : l'aptitude de l'approche à fournir les moyens nécessaires aux concepteurs pour exprimer leurs besoins métiers et leurs besoins de correction et de fiabilité,
- formalisme : l'aptitude à fournir un cadre formel de conception pour les workflows transactionnels tout en assurant un degré de complexité minimal.

3.3.4.1 Positionnement des approches existantes

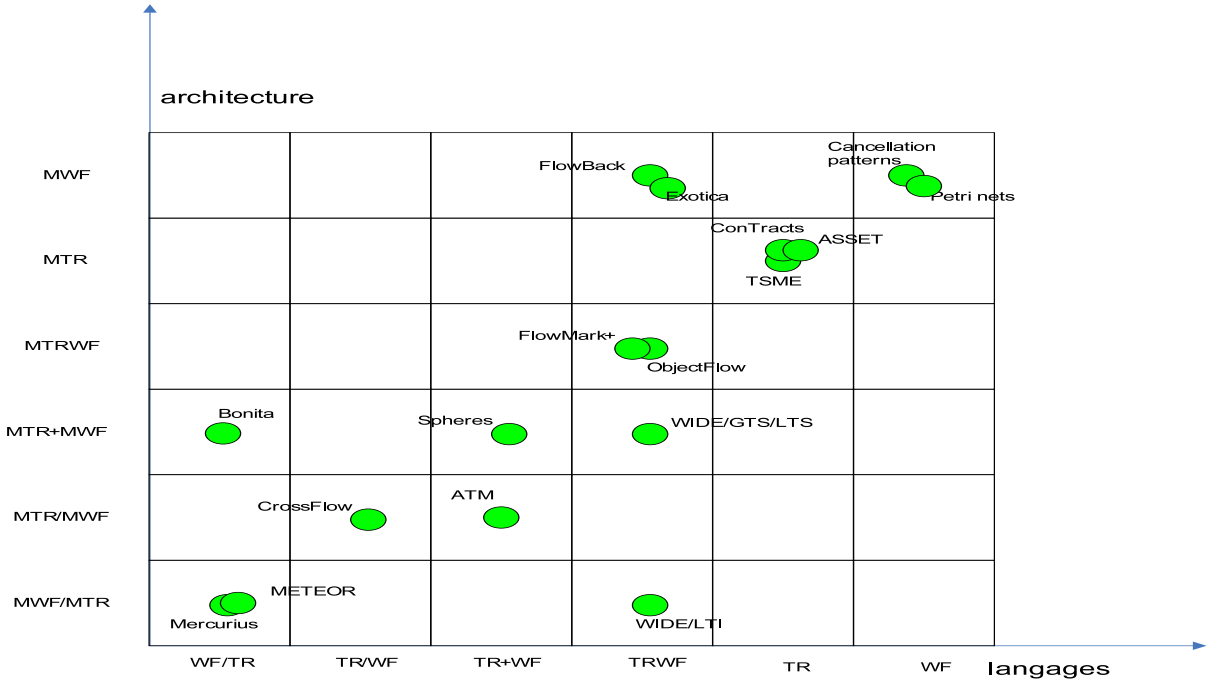
Dans la figure 3.3, nous avons classé un certain nombre des exemples existants de workflows transactionnels présentés dans les sections précédentes selon la conception et l'architecture adoptées par ces exemples. Plusieurs de ces approches peuvent partager la même architecture ou conception sans se confondre totalement. En effet, des points aussi importants que la sémantique, la syntaxe et le domaine d'application du modèle transactionnel ou du modèle de workflow utilisé peuvent les séparer et les rendre très différents même s'ils partagent une conception ou/et une architecture.

Dans la figure 3.3, des propositions de modèles de workflows transactionnels existants ont été positionnées dans une grille bi-dimensionnelle. L'axe horizontal de la grille correspond à l'aspect conceptuel de la classification, l'axe vertical à l'aspect architectural. Les approches situées sur la diagonale de la grille ont des aspects conceptuels architecturaux appartenant à la même classe. Clairement, plusieurs approches appartiennent à cette catégorie.

3.3.4.2 Expressivité et formalisme dans les workflows transactionnels

Nous procédons, dans cette section, par une première comparaison des classes concernant les critères d'expressivité et de formalisme simple. Nous avons analysé la capacité de chaque classe, qu'elle soit conceptuelle ou architecturale, son degré d'expressivité pour répondre aux besoins sémantiques des utilisateurs. Notre analyse (*c.f.* figure 3.4) est basée sur la grille de classification présentée dans la section précédente qui nous propose un moyen efficace pour raisonner au dessus des classes.

Figure 3.3 Positionnement conceptuelle et architecturale.



Les cases grises de la grille de la figure 3.4 concernent les approches qui manquent d'expressivité du fait qu'elles concernent uniquement les classes conceptuelles ou architecturales des modèles conceptuels ou architecturaux simples **TR**, **WF**, **MTR** et **MWF**. Ces classes se caractérisent par un degré d'intégration élevé qui pénalise l'expressivité du modèle bien qu'elle en améliore la cohérence. L'expressivité limitée est un inconvénient clair des classes de **WF** et de **TR**, car la sémantique possible des aspects transactionnels est limitée par les primitives disponibles de workflows et vice versa. Quant aux modèles séparés, la séparation des intérêts entre le modèle de workflow et le modèle transactionnel favorise leur expressivité.

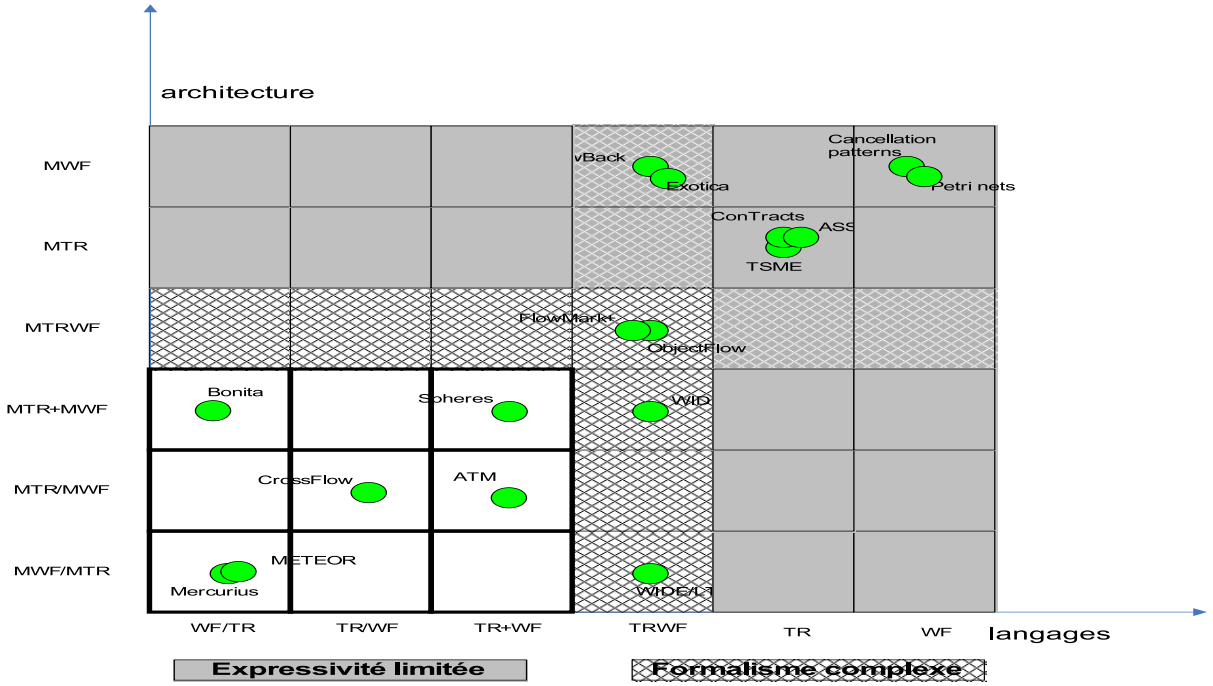
Les cases hachurées concernent les classes conceptuelles et architecturales **TRWF** et **MTRWF** qui se caractérisent par un formalisme et une sémantique complexe. La complexité du formalise est dûe à l'intégration des modèles de workflow et des modèles transactionnels ou de l'architecture intégrée en triangle (*c.f.* figure 3.2.C) qui rendent le formalisme conceptuel et architectural assez complexe et contigu.

Ainsi par déduction de l'analyse précédente, les cases à traits en gras représentent les classes qui répondent au mieux aux besoin de formalisme simple et d'expressivité riche. Nous pouvons constater que ces cases appartiennent à la fois aux classes conceptuelles séparées et aux classes architecturales aux modules séparés.

3.3.4.3 Flexibilité, correction et sémantique dans les workflows transactionnels

Dans cette section, nous traitons le degré de flexibilité qui définit la liberté laissée aux concepteurs pour spécifier leurs besoins tant au niveau de la structure de contrôle par le choix du modèle de workflow qu'au niveau du degré de correction exigé par le choix du modèle transactionnel. Nous nous intéressons à l'aptitude des différentes classes conceptuelles et architecturales à y ré-

Figure 3.4 Expressivité et formalisme dans les workflows transactionnels.



pondre. Finalement, nous identifions les classes conceptuelles et architecturales qui répondent au mieux au besoin d'une sémantique de procédés métiers assez riche. Comme nous l'avons mentionné précédemment, notre analyse se base sur la grille de classification présentée dans la figure 3.3.

Les classes architecturales **MWF**, **MTR** et **MTRWF** signalées par les cases bleues dans la figure 3.5 manquent de flexibilité à cause de la caractéristique mono-module. En effet, elles imposent l'utilisation d'un module unique, ce qui réduit le degré de flexibilité pour le concepteur.

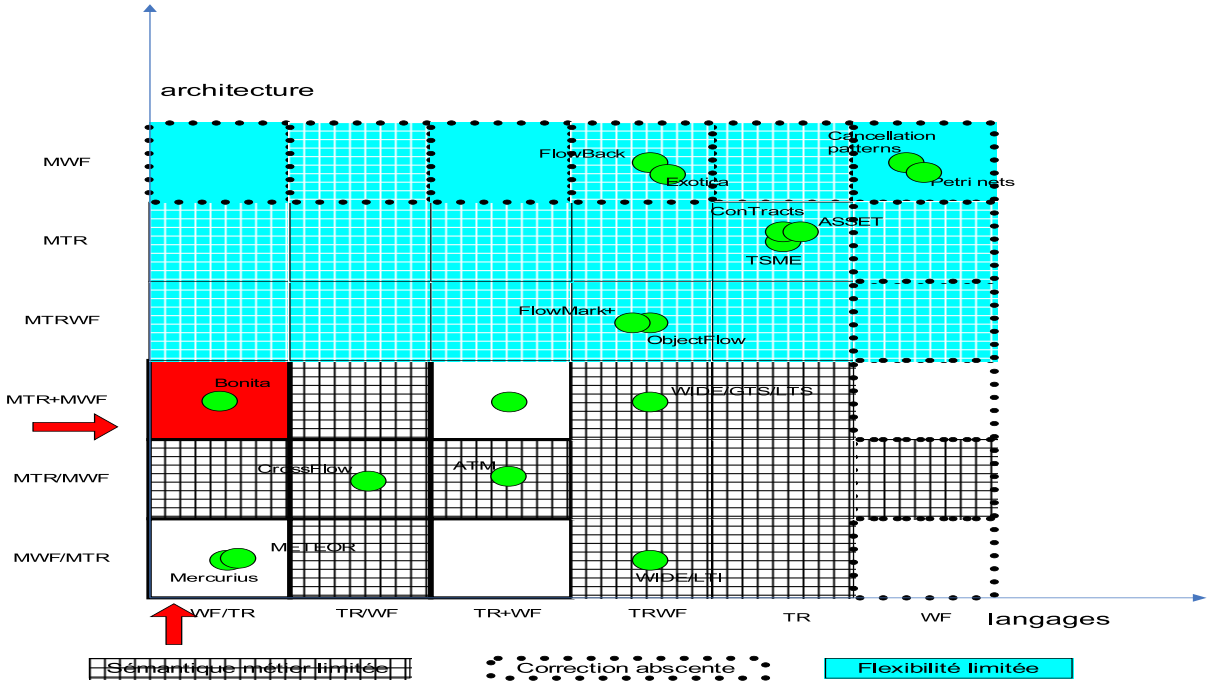
Les cases hachurées ne proposent qu'une sémantique de procédés métiers réduite ou inexistante. En fait, ces cases concernent les classes qui ne font pas du module ou du langage de workflow leur composante de base. Seulement l'intersection des classes conceptuelles **WF** et **TR**+**WF**, et des classes conceptuelles **MWF** et **MWF**+**MTR** offre la garantie qu'on utilise un module et un langage de workflow comme composante principale du workflow transactionnel.

Les cases dont les traits sont pointillés désignent les classes de workflows transactionnels **WF** et **MWF** qui ne fournissent pas ou peu de fonctions de correction, cela est du à l'absence d'un module transactionnel et d'un langage transactionnel.

En conclusion, nous visons à travers cette analyse le choix des classes qui peut nous offrir le plus de flexibilité, d'expressivité, de correction et de sémantique de procédés métiers sans formalisme trop complexe. Quatre cas de la figure 3.5 répondent à ces critères (les cases à traits gras et non hachurés). Mais pour optimiser l'expressivité de la sémantique des procédés métiers, notre choix va se porter sur la classe de langage **WF**/**TR** car elle se base plus sur un langage procédé que la classe **TR**+**WF**. En plus, nous avons fait le choix de la classe architecturale **MTR**+**MWF** qui offre plus de flexibilité que la classe **MWF**/**MTR**. Ce choix est désigné par une case rouge dans la figure 3.5 qui est en fait l'intersection des classe **WF**/**TR** et **MWF**/**MTR**.

Ce choix sera par la suite repris dans le chapitre suivant pour la définition du modèle de

Figure 3.5 Flexibilité, correction et sémantique dans les workflows transactionnels.



workflow transactionnel que nous avons spécifié. Ce modèle de workflow transactionnel peut être aisément implanté dans le système de gestion de workflow Bonita [MC03] développé dans notre équipe de recherche. En effet, le SGWF Bonita propose des mécanismes de modélisation appelés «hooks» qui permettent, entre autres, de spécifier un comportement transactionnel comme un raffinement du langage de description du modèle workflow. Ainsi, Bonita s'inscrit dans la classe conceptuelle **WF/TR**. Par ailleurs, l'architecture de Bonita se caractérise par une séparation entre le module de workflow et le module transactionnel. Concrètement, le module transactionnel dans Bonita peut être spécifié indépendamment du module de workflow dans une couche architecturale séparée. Par conséquent, Bonita s'inscrit dans la classe architecturale **MTR+MWF**. Notons que plus de détails concernant le SGWF Bonita sera apporté dans le chapitre 6 de mise en œuvre.

3.4 Re-ingénierie des procédés

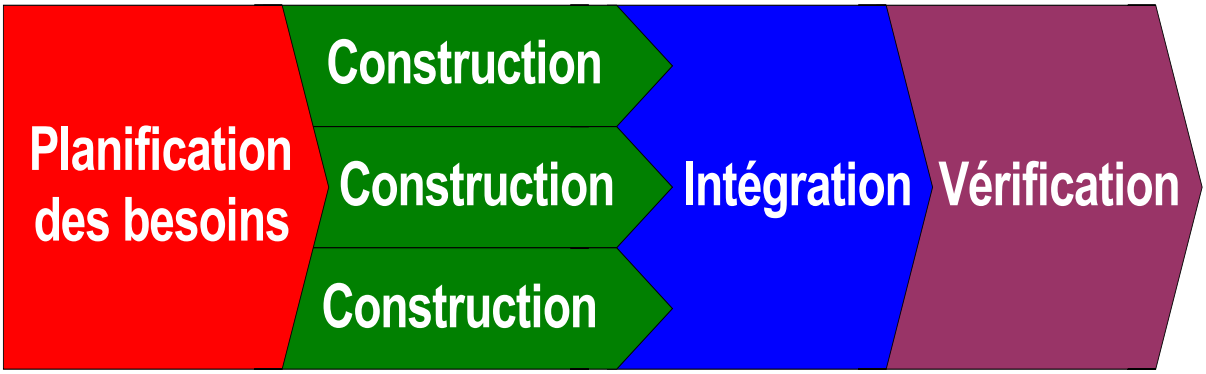
Inconstablement, la conception des procédés métiers a suscité une très grande attention pendant les deux dernières décennies. La manière dont les procédés métiers sont structurés a un grand impact sur le coût, et la qualité de leurs produits. Beaucoup de littératures scientifiques [Han94; JL96; MKK+96] sont consacrées à la présentation et à la discussion des outils de conception spécifiques. La majorité des outils identifiés dans [KT97] se concentrent sur la modélisation et la conception d'un procédé métier.

Dans cette section, nous présentons les différentes approches de conception et les techniques d'aide à la conception des procédés métiers dans les systèmes de gestion de workflows. Nous incluons dans cette vue d'ensemble des méthodes qui peuvent être utilisées pour la vérification, l'évaluation, le diagnostic, la découverte et la re-conception des workflows.

3.4.1 Méthodologie de conception classique des procédés métiers

Dans cette section, nous présentons la méthodologie de conception classique des procédés métiers qui est la plus commune des méthodologies de conception de procédés métiers. Elle est régie par un ensemble de principes et une philosophie commune décrite par l'ensemble d'étapes illustrées dans le schéma 3.6. Quatre étapes successives composent cette méthodologie : la planification des besoins, la construction du schéma du workflow, l'intégration et finalement la vérification du modèle conçu.

Figure 3.6 Approche classique de conception de workflows



La phase d'analyse et la planification des besoins constitue la première étape qui consiste à analyser la situation actuelle, et en particulier les besoins et le produit du procédé métier à concevoir afin de définir les objectifs que le schéma de workflow doit atteindre. Les clients fournissent cette entrée qui est reprise par les concepteurs sous la forme de spécifications. Ces objectifs peuvent être quantifiés pour être repris par la suite dans la dernière phase comme outils de mesure pour la vérification des performances du workflow conçu. Ainsi, durant la phase de planification des besoins, les directives pour les différentes fonctionnalités et les résultats attendus du procédé sont définis.

En se basant sur les spécifications de la phase d'analyse et la planification des besoins, la deuxième étape de construction du workflow est lancée. Pendant cette étape de construction, les différentes spécifications du procédé sont mises noir sur blanc sous la forme d'un schéma de workflow en utilisant l'outil d'édition du système de gestion de workflow choisi pour l'implantation du procédé métier. Les caractéristiques du workflow conçu sont élaborées interactivement dans des ateliers communs de conception à travers des prototypes qui peuvent être sujets de tests par les concepteurs ou des utilisateurs potentiels. Notons qu'on peut être amené à diviser l'étape de construction en plusieurs étapes parallèles. En effet, le procédé est souvent trop grand pour être conçu dans sa totalité. Par conséquent, il peut être utile de développer le workflow dans un certain nombre d'étapes séparées. Chaque étape se termine avec la livraison d'un nouveau schéma de workflow qui est une amélioration/extension du précédent.

Ces différentes versions sont par la suite intégrées dans la troisième étape d'intégration. Cette étape peut se faire dans le cadre d'une collaboration étroite entre les concepteurs du workflow, d'une part, et ses utilisateurs, d'autre part, sous la forme d'un développement commun. Les utilisateurs peuvent ainsi soumettre des amendements aux propositions des concepteurs pour coller au mieux aux objectifs définis dans la première étape. Mais l'organisation d'une telle coopération reste assez **compliquée**, ce qui engendre un manque d'**efficacité**.

Avant que le workflow conçu soit mis en pratique, il est indispensable de vérifier, d'une part, que le modèle conçu ne contient pas d'erreurs conceptuelles ou d'erreurs sémantiques et de s'assurer, d'autre part, que l'ensemble des objectifs définis initialement est atteint dans la pratique. La dernière étape de vérification analyse le schéma de workflow conçu et s'assure qu'il retranscrit fidèlement et sans erreurs les spécifications de la première phase. Cette étape peut inclure des techniques de vérification et d'analyse pour parfaire un diagnostic qui peut détecter des problèmes de correspondance conceptuelle. Ces techniques permettent en particulier d'expérimenter, de mesurer et d'évaluer les performances en utilisant des métrique prédéfinies pour assurer que l'ensemble des objectifs définis initialement est atteint dans la pratique. Van der Aalst [vdA98], Oberweis et autres [OSS+97] et Benatallah et autres [BCWH+03; CWBH+03] identifient des techniques d'évaluation théoriquement prouvées pour l'amélioration des workflows conçus.

Cette étape comporte aussi la vérification de l'exactitude syntaxique d'un modèle de workflow. Contrairement au contexte des langages de programmation, où la syntaxe se rapporte seulement au langage, elle incorpore dans sa notion l'exactitude syntaxique de la structure et du comportement du schéma de workflow. Particulièrement, quand un modèle de workflow devient grand (parfois des centaines d'activités), il est difficile pour les concepteurs humains de contrôler le modèle complet. Un grand nombre d'outils dédiés, comme par exemple l'outil ExSpect [vHSV89; vdAdCG+00], Woflan [VvdA00; VBvdA01], ont été développés à cette fin.

Cette vérification constitue une étape cruciale dans le cycle du vie du procédé. Elle implique l'exactitude syntaxique du modèle. Bien que les spécifications de la phase initiale soient la source appropriée pour dériver ce qui devrait être fait, des fausses interprétations ou des utilisations inexactes peuvent être la cause d'une conception défectueuse de workflow. Une discussion plus détaillé de cette issue est donnée dans [vdA00a].

Du point de vue du développement de système, il est important de valider une conception de procédé avant l'exécution du workflow. Il est bien connu que les erreurs de conception qui sont trouvées tard dans le projet sont très coûteuses à corriger. On a estimé que dans le développement de logiciel une erreur de conception trouvée respectivement pendant la programmation, le test, et la maintenance est 3, 10 et 100 fois plus coûteuse que si elle est trouvée pendant la conception. Cependant ces techniques **manquent de mécanismes de correction** permettant d'optimiser la structure du workflow conçu. En plus ces techniques se caractérisent par une approche **statique**, du fait qu'elles analysent un modèle conçu figé ou **rigide** et ne traitent pas de l'évolution ou du changement dû à l'utilisation future du workflow.

3.4.2 Vers une nouvelle méthode de conception rétro active des workflows

Des méthodes de conception différentes existent déjà ; Alors, pourquoi développer une «nouvelle» méthode ? Les méthodes de conception de workflows attachent de l'importance à la manière avec laquelle le workflow se présente et interagit avec ses utilisateurs. Une méthode pour développer un système de workflows devrait donc plus être portée sur le procédé métier. Les «utilisateurs» doivent être impliqués autant que possible dans la conception des procédés. Ceci signifie que la conception du procédé sera améliorée, par une évaluation et une révision continue jusqu'à ce qu'elle soit satisfaisante.

Le point de départ d'une conception ou d'une nouvelle conception d'un procédé métier constitue un critère fondamental pour toute étude critique et comparative entre méthodologies de conception. Principalement on distingue deux possibilités :

- le procédé est conçu à partir de zéro ;
- le procédé est conçu en prenant le procédé existant comme point de départ.

Il y a une discussion considérable dans la littérature entre la première et la deuxième alternative [OS99]. La méthodologie classique que nous venons de présenter dans la section précédente adopte une approche à feuilles blanches, c.à.d, le procédé est conçu à partir de zéro. Les adversaires de l'approche à feuille blanche identifient trois inconvénients principaux, les menant à préconiser l'utilisation du procédé existant comme point de départ, qui sont :

- L'approche à feuille blanche est incapable de bénéficier de la connaissance et de l'expérience des conceptions ultérieures ni des utilisations du workflow qui se sont accumulés avec le temps. Ainsi elle risque de reproduire les erreurs du passé [MK94; PR95; OS99]
- Les utilisateurs auront du mal a se familiariser avec un nouveau schéma de workflow qui ne tient pas compte les ressemblances avec les versions antérieures [PR95].
- En concevant un procédé complètement à partir de zéro la dimension du problème de la re-conception n'est pas appréciée à sa juste valeur [MK94; PS94; OS99].

Par ailleurs, La W*f*MC définit le workflow comme : «l'automatisation complète ou partielle d'un procédé métier .... selon un ensemble de règles procédurales» [Coa96] et définit le système de gestion d'un workflow comme «un système qui définit, crée et gère l'exécution des workflows par l'utilisation d'un logiciel qui est capable d'interpréter la définition d'un procédé» [Coa96], mettant ainsi l'accent sur la phase d'exécution qui est considérée comme la finalité de la phase de conception. En effet, une conception ne peut être correcte que si sa phase d'exécution fonctionne correctement. Cependant, on peut observer que l'intérêt de la méthodologie classique pour une phase d'exécution comme outil de support pour une conception ou une re-conception n'a été que peu ou pas considéré.

Après plusieurs décennies d'automatisation, beaucoup d'organismes sont arrivés à la conclusion qu'il en faut beaucoup plus pour réaliser des améliorations réelles. Une approche radicale qui exploite le savoir acquis des expériences passées est donc exigé pour obtenir un plus grand rendement. Ainsi, une méthodologie de conception qui prend l'expérience acquise dans la phase d'exécution du **procédé existant comme point de départ** d'une re-conception semble la réponse naturelle pour pallier ces inconvénients et coller au mieux aux besoins.

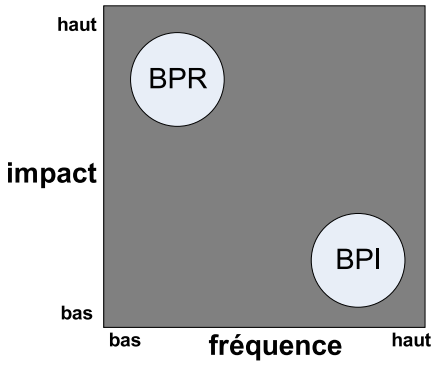
Le fait que nous parlons ici d'une nouvelle méthode ne veut pas dire que nous souhaitons complètement «réinventer la roue». Ceci peut être considéré comme une suite continue de la phase de diagnostic avec l'objectif d'identifier des erreurs ou des oublis conceptuels potentiels en se basant sur l'expérience acquise des exécutions précédentes et en la comparant avec les conceptions antérieures et ainsi procéder à des ajustements améliorant ou corrigeant une conception antérieure.

Au contraire de BPR ou les changements sont radicaux, ces améliorations sont habituellement mineures et ponctuelles. Cette approche est identifiée comme un «processus d'amélioration continue» (CPI)[17]. Puisque les changements ne sont pas aussi grands, la fréquence avec laquelle ils peuvent être mis en application est beaucoup plus haute. Le schéma 3.7 illustre le positionnement relatif de CPI et de BPR. L'utilisation de la technologie «workflow» pour la modélisation des procédés métiers a des avantages clairs à cet égard (l'application du «processus d'amélioration continue»). Puisque les définitions de procédés métiers sont établies en termes de paramètres, leur ajustement exige relativement peu d'efforts et ainsi des décisions plus faciles à prendre.

La connaissance nécessaire au CPI est généralement extraite en étudiant la phase d'exécution. Ainsi les procédés existants sont analysés et un diagnostic est fait par la suite pour voir si des problèmes existent. En utilisant cette connaissance, on peut définir des objectifs par lesquels le succès des améliorations peut être mesuré. Cette approche nous permet de vérifier la correction des conceptions d'une façon **dynamique** et **interactive** en se basant à la fois sur des

[17] Le terme anglophone est Contunious Business Improvement

Figure 3.7 BPR versus BPI



exécutions multiples et une interactivité accrue avec les utilisateurs, au contraire de la méthodologie de conception classique qui impose un diagnostic **statique** de la conception. Une fois que le diagnostic a été fait, la phase de re-conception suit. Dans la phase de création, un nouveau schéma est créé pour supporter le procédé nouvellement conçu. Pendant la phase d'exécution, une nouvelle phase opérationnelle est lancée, ce qui nous permet de mener à nouveau une nouvelle phase de diagnostic et d'évaluation qui peut bien justifier le lancement d'un nouveau cycle de re-conception, comportant très probablement des nouvelles améliorations.

3.4.3 Découverte de procédé

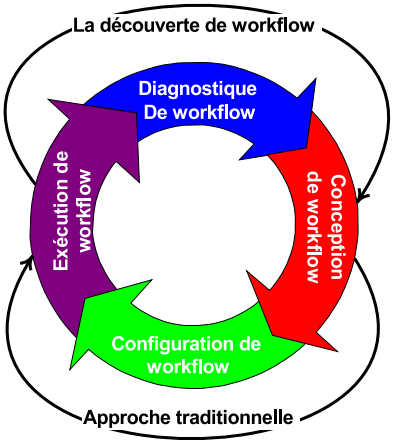
Dans la plupart des systèmes de gestion de procédés, des données relatives à l'exécution sont collectées [vdAvDH+03], par exemple les traces d'exécutions transactionnelles de l'ERP de SAP. Concrètement, l'information rassemblée au temps d'exécution peut être utilisée pour dériver un modèle expliquant les événements enregistrés. Un tel modèle peut être utilisé dans la phase de re-conception. Dans ce cadre, la **découverte de procédé**[18] propose des techniques et des algorithmes inspirés du domaine de fouille de données pour extraire le modèle du procédé à partir de ses traces d'exécutions. Le terme de la découverte de procédé se rapporte à des méthodes pour extraire une description structurée du procédé à partir de ses traces d'exécutions réelles.

Pour comparer la découverte de workflow et l'approche traditionnelle, considérons le cycle de vie de workflow de la figure 3.8. Le cycle de vie de workflow se compose de quatre phases : (a) la conception de workflow, (b) la configuration de workflow, (c) l'exécution de workflow, et (d) le diagnostic de workflow. Dans l'approche traditionnelle la phase de conception est employée pour construire un modèle de workflow. Ceci est typiquement fait par un consultant. Le workflow, à la fin de la conception, est configuré comme indiqué dans la phase de conception. Dans les phases de configuration, on doit traiter des limitations et des particularités du système de gestion de workflow utilisé. Dans la phase d'exécution, des cas (c.à.d, instances de workflow) sont exécutés par le système de workflow comme il a été indiqué dans la phase de conception et réalisé dans la phase de configuration. Dans cette phase, il est possible de rassembler l'information nécessaire au diagnostic qui sera analysée dans la phase de diagnostic.

Au contraire de l'approche traditionnelle, la découverte de workflow permet de comprendre ce qui s'est passé réellement. Son but est de inverser le processus de conception et de rassembler des données à l'exécution pour soutenir la conception et l'analyse de workflow (*c.f.* figure 3.8). Elle commence par la collecte des informations sur les procédés de workflows pendant leurs

[18] Le terme anglophone est «process mining»

Figure 3.8 La découverte de workflow



exécutions. Ces traces d'exécutions sont utilisées pour construire des spécifications de procédés qui modélisent le comportement enregistré. Le défi de la découverte de procédé est de dériver les «bons modèles de workflows» avec le minimum d'informations possibles.

L'idée d'utiliser les traces d'exécutions pour la découverte de procédé a été introduite la première fois par Cook et autres [CW94; CW95; CW98a; CW98b; CW99; CD01] qui ne visent pas vraiment à rechercher un modèle de procédé complet et correct, mais un modèle qui exprime les patrons les plus fréquents dans les traces d'exécutions. Dans leurs premiers travaux [CW94; CW95], ils ont étendu et développé des algorithmes pour découvrir des patrons séquentiels dans le domaine de développement du logiciel. Ces modèles ont été exprimés en tant que machines à états finis. Dans [CW98a; CW98b; CW99], ils étendent leur travaux pour la découverte de modèle de procédés concurrents. Agrawal et autres [AGL98] étaient les premiers à s'intéresser à la découverte de procédé dans un cadre métier. Le modèle de procédé découvert ne montre que les dépendances entre les activités, mais ne donne aucune indication sur la sémantique des points de «jointure» ou de «diffusion». Le travail de Golani et autres [GP03; PG04] est une extension du travail de Agrawal et autres avec la différence principale qu'ils analysent des traces d'exécutions d'activités ayant un événement de début et de fin facilitant ainsi la découverte de la concurrence. L'aspect remarquable de l'approche de Herbst et autres [HK98; Her00b; Her00a; Her00c; HK04] se trouve dans sa capacité d'aborder les tâches dupliquées. Ils présentent un composant d'apprentissage inductif utilisé pour pouvoir supporter l'acquisition et l'adaptation des modèles de procédés séquentiels, généralisant les traces d'exécutions de différentes instances de workflows à un seul workflow couvrant toutes les traces. Schimm [Sch02; Sch04] vise à rechercher un modèle complet et minimal. Schimm découvre un modèle du procédé par la définition d'un ensemble d'axiomes pour appliquer des règles de réécriture au-dessus d'une algèbre de workflow. Les modèles découverts sont des structures en blocs qui ne peuvent pas, cependant, modéliser la synchronisation partielle. Greco et autres [GGP05; GGMS05; GGM$^+$04; GGPS04] visent à découvrir un arbre hiérarchique des modèles de procédé qui décrivent les traces d'exécutions à des niveaux d'abstraction différents. Les auteurs ont une approche en deux étapes. La première étape est une approche descendante qui commence par la découverte d'un procédé racine pour les différents niveaux d'abstraction. Une fois que toutes les racines de l'arbre sont construites, la deuxième étape a lieu. Cette deuxième étape est ascendante. Elle commence aux feuilles de l'arbre découvert et monte jusqu'à la racine de l'arbre pour fusionner les modèles des niveaux d'abstraction inférieurs. Aaslt et autres [vdA00b; WvdA01; WvdA02; MWvdAvdB02;

vdAvD02; vDvdA04; dMWvdA05] ont développé un algorithme de découverte de procédé nommé α-algorithme basé sur les réseaux de Pétri. Les travaux décris par Aaslt et autres traitent la plus part des constructions des réseaux de Pétri et les classifient en démontrant pour quelles classes de modèles leur approche est sûre de fonctionner. Une étude comparative plus exhaustive de ces approches sera donnée dans la section 5.6.

Notons que toutes les approches de découverte de procédé précédentes ont une démarche commune qui prend les traces d'exécutions comme **unique entrée** pour générer le modèle de procédé réel. Ces traces d'exécutions doivent rapporter tout comportement possible du procédé et respecter certaines contraintes sur leur structure concernant principalement l'identification unique des activités et des instances exécutées. Récemment d'autres approches proposent des techniques de découverte de procédé dans le cas où une des caractéristiques précédentes ne soit pas respectée. Par exemple, Jansen-Vullers et autres [JVvdAR06] proposent une solution pour la découverte de procédé à partir de traces d'exécutions amputées de l'information concernant l'identification des activités et des instances exécutées. Leur approche se base sur la fréquence d'exécution d'une transaction et la compare à un modèle de référence. Au contraire des approches de découverte classiques qui utilisent les traces d'exécutions comme unique entrée, leur approche nécessite l'appui d'un modèle de référence. Cette approche n'a pas été l'objet d'une implantation.

Bien que la plupart des techniques de découverte se focalise sur la perspective de la découverte du flux de contrôle, l'exploitation des traces d'exécutions à posteriori est le cœur de plusieurs autres propositions de recherche dans le domaine des système de gestion de procédés. Par exemple, l'analyse du réseau social et ses implications sur la découverte des réseaux sociaux [vdAS04; vdARS05], bien qu'elle soit indépendants des outils et des techniques de découverte de procédé, elle adopte la même approche et exige en plus que chaque événement se rapporte à son acteur. Pour les mêmes raisons que les approches de découverte de procédé, cette approche est une bonne source pour l'amélioration du procédé, mais sous sa forme courante, elle n'est pas applicable pour les systèmes d'information d'entreprises actuels à cause des problèmes de confidentialité des données.

3.4.4 Synthèse

Dans cette section, nous avons présenté en premier une méthodologie classique de conception de procédés. Mais telle qu'elle est définie, cette méthodologie souffre d'une implication déficiente des utilisateurs, tant dans la conception qu'à la vérification dûe à une coopération, qui reste assez **compliquée** à organiser entre d'une part les utilisateurs et les concepteurs, ce qui implique un manque d'**efficacité**. En plus cette méthodologie **manque de mécanismes de correction** permettant d'optimiser la structure du workflow conçu afin de pouvoir assurer son amélioration, et se caractérise par une approche **statique**, du fait qu'elle analyse un modèle conçu **figé** et ne traite pas de l'évolution ou de changement observés après l'utilisation du workflow.

En effet, La méthodologie classique de conception de procédés est influencée par des perceptions, par exemple, les modèles sont souvent normatifs dans le sens qu'ils statuent ce qui «doit» être fait plutôt que de décrire le procédé réel. Une nouvelle orientation a été proposée par la suite pour une méthodologie qui prend comme point de départ les modèles existants et les connaissances acquises suite à la phase d'exécution. Ceci permet une meilleure implication des utilisateurs dans la phase de conception qui nous permet de vérifier la correction des conceptions d'une façon **dynamique**, **interactive** et **continue**, en se basant à la fois sur des exécutions multiples et une interactivité accrue avec les utilisateurs, au contraire de la méthodologie de conception classique qui impose un diagnostic **statique** de la conception.

Dans ce cadre, la découverte de procédé propose un ensemble de techniques d'analyse de

traces d'exécutions des modèles de workflows initialement conçus pour en extraire les modèles de workflows. Ainsi, la découverte de procédé, en utilisant des données liées aux événements réels de l'exécution, peut être utilisée comme outil d'aide pour une méthode de conception plus objective qui se base sur la réalité des traces d'exécutions et corrige les perceptions initiales des concepteurs qui peuvent se relever erronées. Clairement, les techniques de découverte de procédé à partir de leurs traces d'exécutions peuvent être utilisées pour créer une boucle de rétroaction pour adapter le modèle de workflow aux besoins d'évolution et pour détecter les imperfections d'une conception antérieure.

Cependant, les approches existantes de découverte de procédé ne constituent pas, elles seules, un outil de re-conception. En effet, elles s'arrêtent à la découverte du modèle de workflow sans toute fois proposer de mécanismes d'amélioration ou de correction. Mais, il est évident qu'une bonne compréhension du «réel» est essentielle pour n'importe quel effort de re-conception et par conséquent les techniques de découverte de procédé offrent un outil idéal pour une bonne compréhension du «réel».

3.5 Conclusion

Dans ce chapitre, nous avons présenté d'abord les approches principales qui abordent des problèmes connexes au besoin d'un système de workflows correct et cohérent posé par notre problématique : les modèles transactionnels et les systèmes de gestion de workflows classiques. Nous avons montré en particulier leurs limites en étudiant le degré d'intégration des trois principales dimensions que nous avons mises en avant : «flexibilité», «correction» et «procédés métiers». Généralement, les modèles transactionnels assurent un bon niveau de correction au détriment des dimensions de «flexibilité» et «procédés métiers». Les systèmes de workflows assurent un bon niveau de «flexibilité» et sont appropriés au contexte des procédés métiers au détriment de la «correction».

Cette étude nous a conduit à conclure au besoin de combiner ces deux approches dans un seul concept qui est le «workflow transactionnel». Nous avons mené à cet effet une étude des propositions actuelles concernant ce concept selon deux critères : l'aspect conceptuel et l'aspect architectural. Notre étude nous a amené à en distinguer plusieurs niveaux croisés et à désigner le meilleur niveau qui optimise le plus les critères de flexibilité, d'expressivité, de correction et de sémantique de procédés métiers sans formalisme trop complexe. **Dans le le chapitre 4, nous allons proposer un modèle de workflow transactionnel exprimé en calcul événementiel qui suit les recommandations de cette analyse**. Ce modèle fera la base de la solution que nous allons proposer pour répondre au deuxième et principal objectif de notre thèse qui est une conception continue des procédés métiers.

En effet, dans la dernière section de ce chapitre, nous avons abordé les différentes méthodologies de conception de workflow et nous les avons confrontées à notre besoin d'une conception continue et réactive aux besoin d'évolution. Nous avons démontré la nécessité de prendre en compte la connaissance acquise lors de la phase d'exécution précédente. Nous avons présenté la «découverte de procédé» comme un outil d'aide à la conception répondant à ce besoin.

Cependant, nous avons noté que dans l'état de l'art actuel, cette approche n'était pas utilisé comme étant une base d'une possible re-conception. **Dans le chapitre 5, nous allons présenter une nouvelle approche de découverte de workflow qui pallie aux carences des approches existantes et qui fera la base d'une phase corrections et/ou améliorations, en fonction des disparités constatées entre le procédé pré-spécifié et le procédé découvert.**

Chapitre 4

Workflow transactionnel et gestion des échecs d'exécution

Table des matières

4.1 Introduction

Dans ce chapitre, nous présentons notre modèle référence de workflow transactionnel. Rappelons que dans notre travail, nous nous intéressons à assurer des exécutions correctes des workflows en présence des échecs de ses activités composantes. Dans notre modèle, nous distinguons en particulier l'aspect **coordination** (flux de contrôle) et l'aspect **gestion des échecs d'exécution** (flux transactionnel) d'un workflow transactionnel. En effet, d'une part, un workflow peut être

considéré comme une combinaison hybride d'activités automatiques, semi-automatiques et manuelles sous la forme d'un flot d'activités. D'autre part, il peut être considéré aussi comme une transaction structurée où les activités composantes sont les sous-transactions et les interactions sont les dépendances.

Nous présentons un modèle de workflow qui se base sur la combinaison de «patrons» ; un concept simple et fort pour définir des modèles de flux de contrôle d'un workflow. Nous montrons en particulier que cette approche, si elle a pu intégrer les dimensions de flexibilité et fédérer les systèmes de gestion de procédés, a (relativement) ignoré la dimension de «correction» et de gestion d'échecs d'exécution. C'est pourquoi nous décrivons, par la suite, les différents mécanismes de recouvrement adoptés pour tenter de remédier à cette lacune, en intégrant dans notre modèle initial (c.à.d les patrons de workflow) une sémantique transactionnelle afin d'assurer un certain degré de fiabilité [GBG04b; GG05].

La section 4.2 présente une formalisation en calcul événementiel de notre modèle de workflow transactionnel. Nous montrons comment nous modélisons un workflow transactionnel à différents niveaux d'abstraction. Nous distinguons en particulier le `flot de contrôle` (aspect coordination) et le `flot transactionnel` (aspect transactionnel) d'un workflow. Dans la section 4.3, nous présentons le concept de patron de workflow que nous utilisons pour décrire le `flot de contrôle` de notre modèle de workflow transactionnel. La section 4.4 détaille le modèle transactionnel adopté pour décrire le `flot transactionnel` et la sémantique des mécanismes de recouvrement. Nous présentons, en premier, les propriétés transactionnelles que nous considérons et qui découlent du comportement intra-activité, et nous montrons comment ce comportement spécifie ces propriétés transactionnelles. Par la suite, nous exploitons les dépendances transactionnelles inter-activités pour spécifier le flux transactionnel inter-activités. Finalement, nous dévoilons le côté caché des relations sémantiques entre le `flot de contrôle` et le `flot transactionnel` dans la description des mécanismes de recouvrement et de gestion d'échecs d'exécution dans notre modèle de workflow transactionnel.

Il est important de noter que notre modèle référence de workflow transactionnel, qui va être dans le chapitre suivant le modèle de sortie de nos algorithmes de découverte de workflow, est conforme aux standards de la W*f*MC.

4.2 Formalisation des workflows en utilisant le calcul événementiel

Dans cette section, nous montrons comment nous combinons un ensemble d'activités transactionnelles pour spécifier formellement notre modèle de workflow transactionnel. Nous illustrons en particulier comment nous modélisons l'ordonnancement des activités composantes à différents niveaux d'abstraction. Pour la spécification de notre modèle de workflow transactionnel, nous utilisons le calcul événementiel (*c.f.* section 4.2.1) comme formalisme. Au départ, nous montrons comment un workflow transactionnel définit des axiomes sur les actions de ses activités composantes, (*c.f.* section 4.2.2). Ensuite, nous montrons comment ces axiomes permettent d'exprimer à un niveau d'abstraction plus haut des dépendances entre les activités (*c.f.* section 4.2.3). Ces dépendances définissent à leur tour (à un niveau d'abstraction plus haut) le `flot de contrôle` et le `flot transactionnel` du workflow transactionnel (*c.f.* section 4.2.4). Nous reprenons tout au long de cette section notre exemple de workflow de prêt bancaire pour illustrer les différents concepts que nous abordons.

4.2.1 Calcul événementiel

Bien que des propriétés structurelles de base des modèles de procédé aient été largement étudiées, il est remarquable que peu de produits de modélisation de procédé les supportent réellement et formellement. Toutefois une analyse structurelle des modèles de procédés exige une sémantique formelle et claire. Privilégiant une intuitivité et une facilité d'utilisation, une approche pragmatique de modélisation de procédés a été parfois préférée à une autre plus formelle. Cependant, nous mentionnons qu'une sémantique formelle de modélisation de procédés et l'intuitivité et la facilité d'utilisation ne sont pas contradictoires, et les approches de modélisation récentes semblent soutenir cette observation. L'avantage de se référer explicitement au calcul événementiel, qui est un formalisme temporel du premier ordre, est de se doter d'un outil avec une base formelle «solide» permettant de construire des abstractions temporelles à différents niveaux. Ces abstractions sont nécessaires dans le cadre des dépendances d'événements temporels spécifiant l'ordonnancement des activités composant le modèle de workflow.

Cette section récapitule brièvement les fondements du calcul événementiel (plus de détails dans l'annexe A) et présente quelques notions fondamentales. Nous introduisons en particulier les concepts utilisés, et nous définissons la sémantique des événements fournissant un formalisme pour l'expression des clauses comportementales des workflows transactionnels.

Les applications des théories temporelles ont débuté dans les années 1960-70. Elles sont apparues suite à l'application de la logique dans le domaine de l'Intelligence Artificielle. Ces formalismes ont été introduits pour représenter des connaissances temporelles, pour spécifier des domaines d'objets dynamiques ou encore afin de raisonner pour résoudre des problèmes temporelles. Indépendamment de la formalisation, un problème semble être intrinsèque aux théories temporelles lorsqu'elles sont utilisées pour représenter des environnements nécessitant un ordonnancement (planning), à savoir comment décrire des actions afin qu'il soit possible d'en déduire ce que leurs effets impliquent. L'introduction du calcul événementiel [DKRR98; MPW92; Cor02] a permis d'y répondre, car la représentation de celui-ci est assez naturelle et simple pour les domaines d'ordonnancement.

Historiquement, le calcul événementiel a été introduit par Kowalski et Sergot [KS86] comme formalisme logique de programmation pour présenter des événements et leurs effets, particulièrement dans les applications de base de données. Il est un cadre logique dans lequel il est possible d'inférer ce qui est vrai, à certains instants, étant donné un ensemble d'événements et leurs effets et permet de raisonner sur les effets d'actions réelles sur des états locaux. Le calcul est basé sur des axiomes généraux concernant les notions d'événements, leurs propriétés et les périodes de temps où ces propriétés se vérifient. Les événements lancent et/ou terminent les périodes de temps où une propriété se vérifie. Pendant que les événements se produisent, les axiomes généraux infèrent de nouvelles propriétés qui sont jugées vraies dans le nouvel état modélisé, et impliquent l'arrêt d'autres propriétés qui ne sont plus jugées vraies. Le calcul événementiel, largement étudié formellement, est basé sur un modèle d'événements, d'états et de relations de cause à effet et intègre la persistance qui assure qu'une propriété persiste tant qu'un événement ne vient pas l'interrompre.

La formalisation utilisée pour représenter les actions et leurs effets est basée sur la logique du premier ordre. Comme pour tout langage du premier ordre, il faut commencer par choisir l'ontologie sous-jacente c'est-à-dire les symboles de prédicats, les fonctions et les symboles de fonctions. Les concepts importants dans l'ontologie d'un calcul événementiel sont : les actions, les événements, les propriétés et les instants. Les actions et les instants ont leurs significations usuelles. Les actions sont décrites par leurs effets sur le monde et les instants le sont par un nombre réel pour obtenir une représentation du temps continu. Les événements sont des instances

de ces actions à un instant donné. Les propriétés sont des prédicats relationnels dont la valeur, par l'intermédiaire des actions, varie au cours du temps. Cette variation est possible parmi les valeurs d'un ensemble donné. Une propriété peut donc représenter une quantité (Z, R, ...), une valeur de vérité (vrai, faux) ou encore une valeur dans un ensemble quelconque (rouge, bleu, vert).

Un certain nombre de «dialectes» de calcul événementiel ont été présentés depuis le papier originel [KS96a; KS96b; Kow92]. Dans notre travail, nous utilisons la version du calcul événementiel proposée par Shanahan dans [Sha90]. Les concepts utilisés sont inspirés de la logique du premier ordre. Nous distinguons quatre types d'objets que nous regroupons dans des ensembles à savoir :

- $\mathcal{A}ctions$ $\{a, a_i, \ldots\}$: événements ou actions,
- $\mathcal{E}tats$ {variables $f, f_i, \ldots\}$: états, les valeurs des propriétés dépendent du temps.
- $\mathcal{E}stampilles$ {variables $t, t_i, \ldots\}$: des estampilles de temps,
- $\mathcal{O}bjets$ {variables $x, x_i, \ldots\}$: les objets du domaine.
- $\mathcal{P}redicats$ {prédicats $p, p_i, \ldots\}$: les prédicats du domaine.

Comme nous utilisons une version simplifiée du calcul événementiel, seuls les prédicats de base sont décrits ici (*c.f.* tableau 4.1). Les événements et les actions ne sont pas différenciés et les actions seront considérées instantanées. Ce formalisme est largement suffisant pour introduire les concepts de base de notre modèle de workflow transactionnel. En effet, il fournit un cadre pour le raisonnement temporel en utilisant la logique des prédicats du premier ordre. Dans ce cadre il est possible de maintenir une représentation d'un modèle de workflow dynamique à un niveau d'abstraction élevé en se basant sur les axiomes fournis par l'utilisateur. Ces axiomes permettent de modeler des activités séquentielles, concurrentes et synchronisées couvrant le spectre de comportements possibles dans un workflow recensé par la WfMC [Fis00]. L'addition de cette dimension temporelle aux workflows peut améliorer leurs fonctionnements en fournissant des mécanismes formels pour analyser et évaluer différentes propriétés comportementales du workflow.

Prédicat	**Interprétation**
$happens(a, t1, t2)$	Le déclenchement de l'action a s'est produit entre les instants $t1$ et $t2$
$happens(a, t)$	Prédicat simplifié de $happens(a, t1, t2)$ qui se produit à t
$holdsAt(f, t)$	L'état f est vrai à à l'instant t
$initiates(a, f, t)$	L'état f devient vrai après l'action a à l'instant t
$terminates(a, f, t)$	L'état f devient faux après l'action e à l'instant t
$Initially_P(f)$	L'état f est vrai à partir de l'instant initial, c-à-d 0

TAB. 4.1 – Prédicats du calcul événementiel

4.2.2 Modélisation du comportement d'une activité transactionnelle

Un workflow transactionnel compose un ensemble d'activités pour atteindre un objectif commun. Concrètement, un procédé de workflows contient une collection d'activités et l'ordre des invocations ou les conditions sous lesquelles les activités sont invoquées. L'architecture, que nous présentons, adopte une approche événementielle pour assurer l'ordonnancement et la synchronisation entre les activités. Cette approche est définie par la spécification des axiomes décrivant les actions exécutées et leurs effets sur les états des activités assurant la description des exécutions d'instances de workflow.

Un workflow transactionnel implique plusieurs activités transactionnelles, chacune exécutée par un agent (humain ou automatisé). Une activité transactionnelle (*c.f.* définition 4.1) est décrite par un identifiant unique et l'ensemble de ses états potentiels lors de l'exécution, ainsi que les actions déclenchant les transitions entre ses différents états, et les interactions avec l'extérieur (les autres activités).

A chaque activité transactionnelle est associé un diagramme à transitions d'états qui modélise son comportement (transactionnel). Ce diagramme décrit les états possibles par lesquels une activité peut passer durant son cycle de vie et les transitions possibles entre ses états. Ces transitions sont déclenchées par des actions.Nous Notons $\mathcal{AT}$ l'ensemble de toutes les activités transactionnelles.

Nous définissons la fonction *états* : $\mathcal{AT} \longrightarrow \mathcal{P}^{19}(\mathcal{E}tats)$ qui permet de retourner l'ensemble des états d'une AT donnée.

Définition 4.1 (Activité Transactionnelle)
Une activité transactionnelle, at, est un triplet $at = (ID \in \mathcal{O}bjets, E \subset \mathcal{E}tats, A \subset \mathcal{A}ctions)$ où ID est un objet désignant l'identifiant de l'activité, E et l'ensemble de ses états potentiels et A l'ensemble des actions potentielles déclenchant les transitions entre ses états.

L'activation d'une transition est enclenchée à travers une action qui fait transiter l'activité en question d'un état initial vers un nouvel état. Une action at.`act()`[20] d'une activité transactionnelle at initialise un nouvel état et termine un autre dans l'activité (*c.f.* la définition 4.2). Les axiomes sont utilisés pour décrire le changement d'états d'une activité. Dans la suite, nous détaillons l'ensemble des états et des actions respectifs que nous considérons dans notre approche et qui sont conformes aux indications de la W*f*MC.

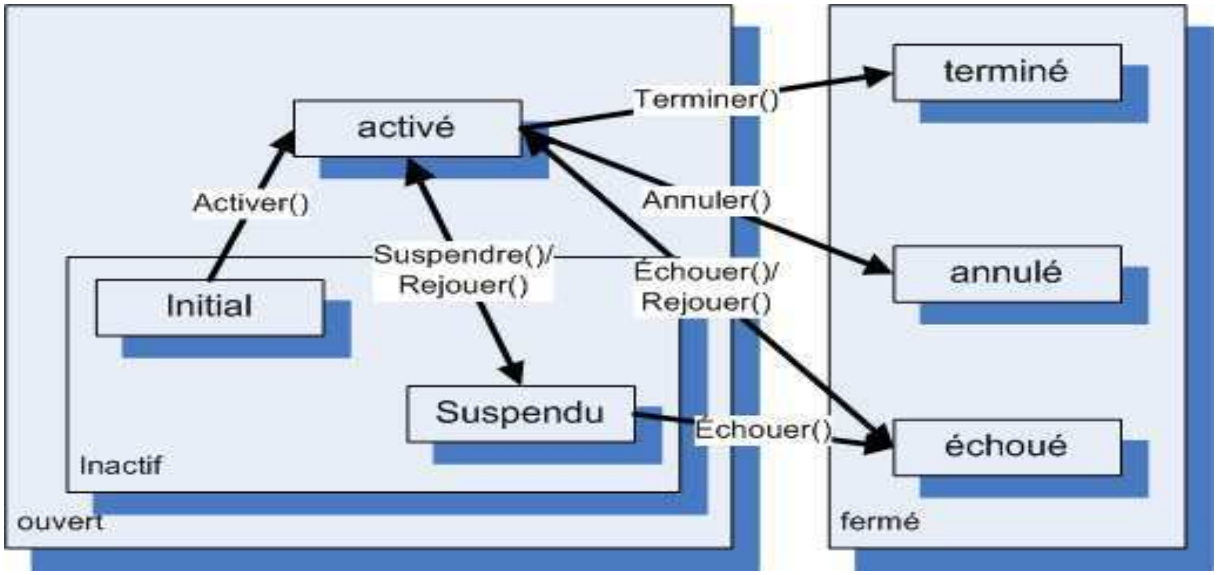
Définition 4.2 (Action d'une Activité transactionnelle)
Soit at une activité transactionnelle, chaque action `act()` de at définit deux axiomes :
– un axiome d'initialisation d'un nouvel état e_1 dans l'activité at : $initiates(at.\texttt{act()}, e_1(at))$
– un axiome de terminaison de l'état courant e_2 dans l'activité at : $terminates(at.\texttt{act()}, e_2(at))$
$e_1, e_2 \in$ *états(at)*

Pour chaque activité transactionnelle at {`initial(`at`)`, `suspendu(`at`)`, `activé(`at`)`, `annulé(`at`)`, `échoué(`at`)`, `terminé(`at`)`} est l'ensemble potentiel des états que nous considérons et {at`.suspendre()`, at`.activer()`, at`.annuler()`, at`.échouer()`, at`.terminer()`, at`.rejouer()`} est l'ensemble des actions potentielles que nous associons aux transitions de at. Dans le diagramme de la figure 4.1, qui est conforme aux spécification de la W*f*MC [Coa96], on distingue deux principaux «super» états `ouvert` et `fermé`. Le «super» état `ouvert` a trois états atomiques `initial`, `suspendu` et `activé`. Les deux premiers états atomiques sont classés comme états inactifs. Le «super» état `fermé` a trois états atomiques `terminé`, `échoué` et `annulé`. Les transitions potentielles entre les états sont décrites par des actions qui permettent de les déclencher. Ces transitions concernent les états de bas niveau (c.à.d atomiques), les transitions entre les états de plus haut niveau peuvent en être déduites facilement. Initialement, à l'initiation d'une instance de workflow, toutes les activités transactionnelles du workflow sont à l'état `initial`. L'action `activer()` permet d'activer une activité et de la faire ainsi passer de l'état

[19] Étant donné un ensemble E, nous notons $\mathcal{P}(E)$ pour désigner l'ensemble des sous ensemble de E.
[20] at. `act()` désigne l'exécution de l'action `act()` sur l'activité at et $e(at)$ désigne l'état e de l'activité at

`initial` à l'état `activé`. Une fois activée, l'exécution de l'activité peut être suspendue ; l'action `at.suspendre()` permet de suspendre l'exécution de *at* et le fait passer ainsi de l'état `activé` à l'état `suspendu`. Elle peut à nouveau être re-activée par l'action `at.rejouer()`. Par la suite, elle peut continuer normalement son exécution ou peut être annulée (durant son exécution). Ainsi, l'action `annuler()` permet d'annuler une activité (en cours d'exécution) et la fait ainsi transiter de l'état `activé` vers l'état `annulé`. Si une activité n'est pas annulée durant son exécution, elle peut soit atteindre ses objectifs et se terminer avec succès ou échouer. Ainsi, l'action `échouer()` permet de marquer l'échec d'une activité et la fait ainsi transiter de l'état `activé` ou `suspendu` vers l'état `échoué` et l'action `terminer()` permet de marquer la terminaison d'une activité avec succès et de la faire ainsi transiter de l'état `activé` vers l'état `terminé`. Finalement, l'action `rejouer()` permet d'activer une activité après un échec ou une suspension et la fait ainsi transiter de l'état `échoué` ou `suspendu` vers l'état `activé`.

Figure 4.1 Diagrammes à transitions d'états des activités transactionnelles.



Quelques systèmes ont un niveau de granulité plus fin que les actions et les états montrés dans figure 4.1, d'autres gardent seulement une partie du diagramme 4.1. D'autre part, l'appellation des types d'états et d'actions peut varier. Comme exemple, considérons le système de gestion de workflows Staffware, il ne décrit que les états `initial`, `abandonné` et `terminé` et omet les états `activé`, `suspendu` et `annulé` .

Par ailleurs, nous distinguons entre les actions inter-activités et les actions intra-activité d'une activité transactionnelle. D'une part, une action intra-activité est activée par l'activité elle-même (l'agent responsable de l'exécuter). Les actions intra-activité que nous considérons dans notre approche sont `terminer()`, `échouer()` et `rejouer()`. D'autre part, une action inter-activités est activée par une entité externe : une autre activité ou toute autre entité externe (agent humain, service d'activation du workflow, événement extérieur, etc.). Les actions inter-activités permettent principalement à une activité d'interagir avec les autres activités du même workflow et de décrire ainsi **la dynamique inter-activités du workflow**. Les actions inter-activités que nous considérons dans notre approche sont : `suspendre()`, `activer()` et `annuler()`.

Nous notons $\mathcal{A}ct\mathcal{I}nter$ et $\mathcal{A}ct\mathcal{I}ntra$ respectivement l'ensemble de toutes les actions inter-

Actions	**Définition du domaine**
	$Initially_P(initial(at))$
`terminer()`	$initiates$(at.`terminer()`,`terminé`(at), t) $\leftarrow HoldsAt$(`activé`(at),t) $\wedge\ Happens$(at.`terminer()`,t) $terminates$(at.`terminer()`,`activé`(at),t) $\leftarrow HoldsAt$(`activé`(at),t) $\wedge\ Happens$(at.`terminer()`,t)
`échouer()`	$initiates$(at.`échouer()`,`échoué`(at), t) $\leftarrow HoldsAt$(`activé`(at),t) $\wedge\ Happens$(at.`échouer()`,t) $terminates$(at.`échouer()`,`activé`(at),t) $\leftarrow HoldsAt$(`activé`(at),t) $\wedge\ Happens$(at.`échouer()`,t) $terminates$(at.`échouer()`,`suspendu`(at)) $\leftarrow HoldsAt$(`suspendu`(at),t) $\wedge\ Happens$(at.`échouer()`,t)
`rejouer()`	$initiates$(at.`rejouer()`,`activé`(at), t) $\leftarrow HoldsAt$(`échoué`(at),t) $\wedge\ Happens$(at.`rejouer()`,t) $terminates$(at.`rejouer()`,`échoué`(at),t) $\leftarrow HoldsAt$(`échoué`(at),t) $\wedge\ Happens$(at.`rejouer()`,t) $terminates$(at.`rejouer()`,`suspendu`(at),t) $\leftarrow HoldsAt$(`suspendu`(at),t) $\wedge\ Happens$(at.`rejouer()`,t)

TAB. 4.2 – Prédicats du domaine de définition d'une activité transactionnelle

activités et l'ensemble de toutes les actions intra-activité de toutes les activités transactionnelles. Nous définissons l'ensemble des fonctions suivantes nécessaires pour la formalisation des divers concepts introduits par la suite.

- Nous définissons la fonction $Inter : \mathcal{AT} \longrightarrow \mathcal{P}(\mathcal{ActI}nter)$ qui permet de renvoyer l'ensemble des actions inter-activités d'une AT.
- Nous définissons la fonction $Intra : \mathcal{AT} \longrightarrow \mathcal{P}(\mathcal{ActI}ntra)$ qui permet de renvoyer l'ensemble des actions intra-activité d'une AT.

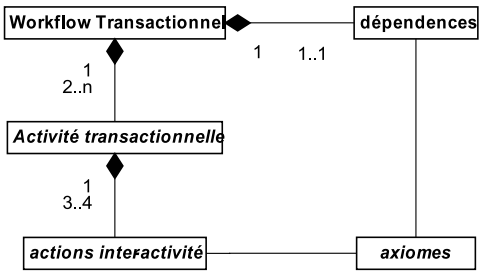
La **définition du domaine** est capturée par un ensemble d'axiomes qui spécifient les clauses *initiates* et *terminates* des actions intra-activité. Cette **définition du domaine** spécifie la dynamique intra-activité qui est décrite initialement puisqu'elle ne dépend pas des événements externes. Ainsi les axiomes des prédicats *initiates* et de *terminates* des actions intra-activité ne dépendent que des états de l'activité ou de l'apparition d'autres actions de la même activité. Concrètement, l'activation de ces actions intra-activité est gérée par l'activité elle même, au contraire des actions inter-activités qui sont activées par une entité externe et qui décrivent la dynamique du workflow en entier à travers l'observation des événements ou des états externes à l'activité traitée. Le tableau 4.2 regroupe des axiomes des actions intra-activité, quant aux axiomes inter-activités, la section 4.2.4 les reprend pour décrire le flux de contrôle et le le flux transactionnel inter-activités.

Les états et les actions potentiels spécifiés dans la **définition du domaine** donnée dans le tableau 4.2 vont être repris partiellement ou globalement par la suite pour décrire les différentes propriétés transactionnelles. Ainsi certaines activités transactionnelles ne peuvent pas reprendre tous les actions et états décrits ci-dessus. Leurs ensembles d'états et d'actions vont dépendre de la nature des propriétés transactionnelles qui les caractérisent et raffinent ces ensembles. Nous reprendrons avec plus de détails la description de ces propriétés transactionnelles et des états et des actions correspondants dans la section 4.2.4.

4.2.3 Dynamique inter-activités du workflow

Pour définir les interactions de ses activités composantes, une activité spécifie des axiomes sur ses actions externes (*c.f.* diagramme UML de la figure 4.2). Ces axiomes spécifient pour chaque activité les conditions sous lesquelles elle sera suspendue, activée, ou annulée. Par exemple, le workflow du prêt bancaire spécifie que l'activité LD sera `activée` après la terminaison des activités VD et RD. Ceci signifie que les axiomes de déclenchement de l'action `activer()` de l'activité LD est la terminaison de l'activité VD et la terminaison de l'activité RD.

Figure 4.2 Dépendances, axiomes et actions dans un workflow transactionnel.



Ainsi, un workflow transactionnel peut être défini comme le couple de l'ensemble de ses activités composantes et l'ensemble des axiomes définis sur leurs actions externes (*c.f.* définition 4.3) .

Définition 4.3 (Workflow Transactionnel : WfT)
Un workflow transactionnel wft est un couple $wft = (WA \subset \mathcal{AT}, Ax \subset \mathcal{A}xiomes)$ où WA est l'ensemble de ses activités transactionnelles composantes et Ax est l'ensemble des axiomes définis par la fonction $\mathcal{A}xm$ qui définit pour chaque action `act()` inter-activités d'une activité composante a, l'axiome relatif à son déclenchement (*c.f.* définition 4.4) :

$$\begin{array}{rccc} \mathcal{A}xm : & \mathcal{A}ctions_{Inter} & \longrightarrow & \mathcal{A}xiome_{wft} \\ & a.\texttt{act()} & \longmapsto & \mathcal{A}xm(a.\texttt{act()}) \end{array}$$

où $\mathcal{A}ctions_{Inter}$ est l'ensemble de toutes les actions inter-activités de toutes les activités composantes de WfT et $\mathcal{A}xiome_{Inter}$ est l'ensemble des axiomes tenant compte seulement des actions générées par les activités composantes de wft.

La fonction $\mathcal{A}xm$ définit pour chaque action inter-activités d'une activité a composante l'axiome relatif au prédicat du déclenchement d'une action `act()` de l'activité a à l'instant t ($happens(a.\texttt{act()}, t)$). Ainsi, la fonction $\mathcal{A}xm$ définit pour chaque prédicat *happens* relatif à une action inter-activités d'une activité composante l'ensemble de prédicats (conditions) susceptibles de la déclencher. Puisque ces conditions sont exclusives, nous les écrivons comme une seule condition sous forme normale disjonctive exclusive.

Ainsi, les axiomes des actions inter-activités d'une activité composante permettent de décrire pour chacune des activités composantes comment elle réagit à la dynamique du workflow, c.à.d. aux actions des autres activités composantes, et comment elle agit sur les comportements des autres activités composantes à travers ces actions. Ainsi, nous distinguons pour chaque activité composante a, un axiome pour chacune de ses actions inter-activités : `activer()`, `suspendre()`, `annuler()`, que nous notons respectivement $\mathcal{A}xiome(\texttt{activer()})$, $\mathcal{A}xiome(\texttt{suspendre()})$ et $\mathcal{A}xiome(\texttt{annuler()})$.

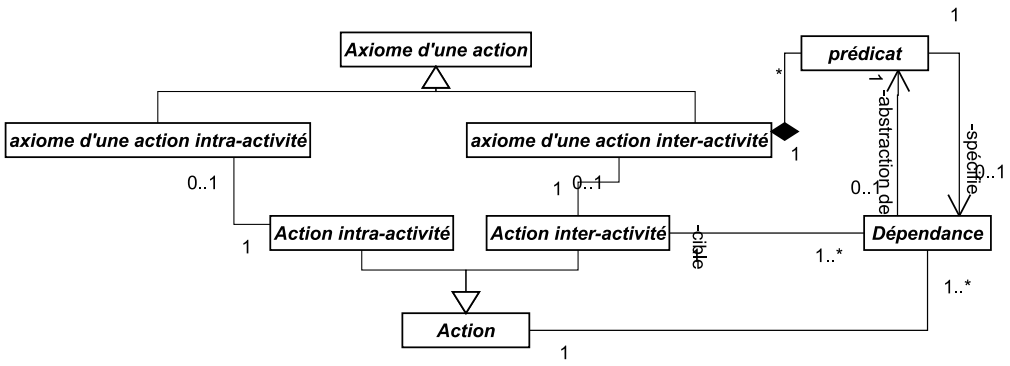
DÉFINITION 4.4 (AXIOME DE LA DYNAMIQUE D'UNE ACTION INTER-ACTIVITÉS)
Soit Wf un workflow, a une activité composante de Wf et ($\texttt{act()} \in \mathcal{A}ct\mathcal{I}nter$)) une action inter-activités de a, l'axiome de $act(a)$ définit pour le prédicat $happens(\texttt{act()}(a), t)$, l'ensemble de conditions susceptibles de l'activer à un instant t. Plus formellement :

$$\mathcal{A}xiome(a.\texttt{act()}) = happens(a.\texttt{act()}, t) \leftarrow \wedge_{i=1..n} P_i, \text{ où } P_i \text{ est un prédicat.}$$

Nous définissons la fonction *prédicats* : $\mathcal{A}xiome_{at} \longrightarrow \mathcal{P}(\mathcal{P}\text{rédicats})$ qui permet de retourner l'ensemble des prédicats P_i.

Par rapport à notre exemple, le workflow spécifie que l'activité VD sera annulée quand l'activité VSC échoue. Ceci signifie que $\mathcal{A}xm$(VD.`annuler()`) = $happens$(VD.`annuler()`,t) ← $happens$(VSC.`échouer()`,t). De même, le workflow spécifie que l'activité MJR est `activée` après la terminaison de ER. Ceci signifie que $\mathcal{A}xiome$(MJR.`activer()`)= $happens$(MJR.`activer()`,t) ← $happens$(ER.`terminer`,t).

Figure 4.3 Vue d'ensemble : Niveaux d'abstractions actions, axiomes et dépendances



Les axiomes expriment à un niveau d'abstraction plus haut des relations entre les activités composantes sous forme de dépendances (*c.f.* diagramme UML de la figure 4.3). Ces dépendances expriment comment les activités sont couplées et comment le comportement de certaines activité(s) influence le comportement d'autre(s) activité(s). D'une façon générale une dépendance d'une activité a_1 vers une activité a_2 existe, s'il existe une action $\texttt{act}_1$ de a_1 et une action externe $\texttt{act}_2$ de a_2 tel que le déclenchement de $\texttt{act}_1$ (c.à.d $happens(a_1.\texttt{act}_1())$) fait partie des prédicats de l'axiome de $\texttt{act}_2()$.

4.2.4 Flot de contrôle et flot transactionnel

Les dépendances permettent d'exprimer différents types de relations (*succession, alternative, annulation, etc.*). La définition 4.5 donne une définition générale d'une dépendance qui permet d'exprimer tout type de relations entre deux activités transactionnelles. Dans notre approche, nous distinguons entre deux catégories de dépendances : les dépendances d'exécutions normales et les dépendances d'exécutions exceptionnelles.

Les dépendances d'exécutions «normales» (par dualité «exceptionnelles») capturent seulement des dépendances des actions inter-activités relatives à des instances de workflows exécutées sans «exceptions» (*c.f.* définition 4.6). Nous définissons une exécution d'instance de workflow nor-

DÉFINITION 4.5 (DÉPENDANCE D'UNE ACTION $a_1.\texttt{act}_1()$ VERS UNE ACTION $a_2.\texttt{act}_2()$)
Soit Wf un workflow, a_1 et a_2 deux activités composantes de Wf, $\texttt{act}_1()$ une action relative à a_1 et $\texttt{act}_2()$ une action relative à a_2, une dépendance de $\texttt{act}_1()$ vers $\texttt{act}_2()$, est un prédicat noté $dep(a_1.\texttt{act}_1(), a_2.\texttt{act}_2())$ qui est vrai ***ssi*** l'apparition de $a_1.\texttt{act}_1()$ peut entraîner l'apparition $a_2.\texttt{act}_2()$. Plus formellement :

$$\exists dep(a_1.\texttt{act}_1(), a_2.\texttt{act}_2()) \Leftrightarrow \exists happens(a_1.\texttt{act}_1(), t) \in prédicats(\mathcal{A}xiome(a_2.\texttt{act}_2()))$$

Nous appelons $a_1.\texttt{act}_1()$ (respectivement $a_2.\texttt{act}_2()$) l'action source (respectivement cible) d'une dépendance $dep(a_1.\texttt{act}_1(), a_2.\texttt{act}_2())$.
Nous notons $\mathcal{D}$épendance, l'ensemble des dépendances.

male comme une exécution où toutes les activités composantes ont atteint avec «succès» l'état `terminé` sans qu'elles soient `suspendues`, `échouées` ou `annulées` et où les axiomes dépendent seulement de la terminaison d'une ou plusieurs autres activités. Ainsi, ces dépendances relatent les interactions entre les activités dans des instances de workflows exécutées sans «exceptions» (c.à.d sans l'observation des états `suspendu` ou `échoué` ou `annulé`).

DÉFINITION 4.6 (DÉPENDANCE D'EXÉCUTIONS NORMALES)
Une dépendance de flux de contrôle de l'activité a_1 vers a_2, notée $depNrm(a_1, a_2)$, existe ***ssi*** une dépendance d'actions existe entre une action source ($a_1.\texttt{terminer()}$) et une action destination ($a_2.\texttt{activer()}$). En termes d'états et d'axiomes, $depNrm(a_1, a_2)$ existe ***ssi*** l'état `terminé` de a_1 est un prédicat de l'axiome d'activation de a_2. Plus formellement et par référence à la définition 4.5 :

$$depNrm(a_1, a_2) \stackrel{\text{def}}{=} dep(a_1.\texttt{terminer()}, a_2.\texttt{activer()}).$$
$$\exists\ depNrm(a_1, a_2) \Longleftrightarrow happens(a_1.\texttt{terminer()}, \text{t}) \in prédicats(\mathcal{A}xm(\texttt{activer().act()}_2))$$

L'axiome qui spécifie cette dépendance d'exécutions normales est défini comme axiome d'activation normale.

Le tableau 4.3 montre comment une dépendance d'exécutions normales $depNrm(a_1, a_2)$ définissant un axiome d'activation normale de a_2 par a_1 enrichit son **domaine de définition** initial.

Actions	**Prédicats d'activation normale**
$a_1.\texttt{terminer()}$	$Happens(a_2.\texttt{activer()}, \text{t2}) \leftarrow$ $Happens(a_1.\texttt{terminer()}, \text{t1})\ ;\ \text{t1}<\text{t2}$
$a_2.\texttt{activer()}$	$initiates(a_2.\texttt{activer()}, \texttt{activé}(a_2), \text{t2}) \wedge$ $terminates(a_2.\texttt{activer()}, \texttt{initial}(a_2), \text{t2}) \leftarrow Happens(a_2.\texttt{activer()}, \text{t2})$

TAB. 4.3 – Prédicats de dépendance d'exécutions normales d'une activité transactionnelle

Après un échec ou une exception d'exécution d'une activité (c.à.d l'exécution des actions **suspendre()** ou **échouer()** ou **annuler()**), l'instance de workflow peut se trouver dans un état «instable» ou «exceptionnel». Des actions inter-activités définies à travers les dépendances d'exécutions exceptionnelles (*c.f.* définition 4.7) peuvent alors être mises à l'œuvre par les entités

DÉFINITION 4.7 (DÉPENDANCE D'EXÉCUTION EXCEPTIONNELLE)
Une dépendance d'exécutions exceptionnelles de l'activité a_1 vers a_2, notée $depEx(a_1, a_2)$, existe ***ssi*** une dépendance d'actions existe entre une action source d'échec (a_1.`échouer()`), de suspension (a_1.`suspendre()`), ou d'annulation (a_1.`annuler()`) et tout autre type d'action destination (a_2.`act()`) en vue d'engager un mécanisme de recouvrement après l'échec de l'activité a_1. En termes d'états et d'axiomes, $depEx(a_1, a_2)$ existe ***ssi*** les prédicats de déclenchement de l'action **échouer()** ou **suspendre()** ou **annuler()** de a_1 est un prédicat de l'axiome de la l'action a_2.`act()`. Plus formellement et par référence à la définition 4.5 :

$$depEx(a_1, a_2) \stackrel{\text{def}}{=} dep(a_1.\texttt{échouer()}, a_2.\texttt{act()}) \vee dep(a_1.\texttt{annuler()}, a_2.\texttt{act()}) \vee dep(a_1.\texttt{suspendre()}, a_2.\texttt{act()}).$$

$$\exists\, depEx(a_1, a_2) \Longleftrightarrow \exists\, happens(a_1.\texttt{échouer()}, t) \vee happens(a_1.\texttt{annuler()}, t) \vee happens(a_1.\texttt{suspendre()}, t) \in prédicats(\mathcal{A}xm(a_2.\texttt{act()}))$$

Selon l'action inter-activités `act()` nous distinguons 3 catégories d'axiomes qui définissent cette dépendance d'exécutions exceptionnelles :
- axiome d'activation alternative si `act()` = `activer()`
- axiome d'annulation si `act()`= `annuler()`
- axiome de suspension si `act()` = `suspendre()`

Actions	**Prédicats d'activation alternative**
a_1.`act()` `act()`=`échouer()`∨`annuler()`∨`suspendre()`	$Happens(a_2.\texttt{activer()}, t1) \leftarrow$ $Happens(a_1.\texttt{act()}, t1)$; t1<t2
a_2.`activer()`	$initiates(a_2.\texttt{activer()}, \texttt{activé}(a_2), t2) \wedge$ $(terminates(a_2.\texttt{activer()}, \texttt{initial}(a_2), t2) \vee$ $terminates(a_2.\texttt{activer()}, \texttt{suspendu}(a_2), t2)) \leftarrow$ $Happens(a_2.\texttt{activer()}, t2)$
Actions	**Prédicats d'annulation**
a_1.`act()` `act()`=`échouer()`∨`annuler()`∨`suspendre()`	$Happens(a_2.\texttt{annuler()}, t1) \leftarrow$ $Happens(a_1.\texttt{act()}, t1)$; t1<t2
a_2.`annuler()`	$initiates(a_2.\texttt{annuler()}, \texttt{annulé}(a_2), t2) \wedge$ $terminates(a_2.\texttt{annuler()}, \texttt{activé}(a_2), t2) \leftarrow$ $Happens(a_2.\texttt{annuler()}, t2)$
Actions	**Prédicats de suspension**
a_1.`act()` `act()`=`échouer()`∨`annuler()`∨`suspendre()`	$Happens(a_2.\texttt{suspendre()}, t1) \leftarrow$ $Happens(a_1.\texttt{act()}, t1)$; t1<t2
a_2.`suspendre()`	$initiates(a_2.\texttt{suspendre()}, \texttt{suspendu}(a_2), t2) \wedge$ $terminates(a_2.\texttt{activer()}, \texttt{activé}(a_2), t2) \leftarrow$ $Happens(a_2.\texttt{suspendre()}, t2)$

TAB. 4.4 – Prédicats de dépendances d'exécutions exceptionnelles d'une activité transactionnelle

externes (programmateur, intervention humaine, etc.) pour permettre à l'instance du workflow d'agir afin de «réguler» ou de stabiliser son état. Le but est de remettre le workflow de nouveau dans un état sémantiquement stable ou normal.

Le tableau 4.4 montre comment une dépendance d'exécutions exceptionnelles $depEx(a_1, a_2)$ définissant des axiomes d'activation alternative, ou/et d'annulation ou/et de suspension de a_2 par a_1 enrichit son **domaine de définition** initial.

Les dépendances d'exécutions normales et les dépendances d'exécutions exceptionnelles expriment à un niveau d'abstraction plus haut respectivement le `flot de contrôle` et le `flot transactionnel` d'un WfT (*c.f.* diagramme UML de la figure 4.5).

Flot de contrôle

Le `flot de contrôle` d'un WfT définit un ordre partiel entre les activations de ses activités composantes. Intuitivement, le `flot de contrôle` d'un WfT est défini par l'ensemble de ses dépendances d'exécutions normales. Formellement, nous définissons un `flot de contrôle` comme un WfT dont les seules dépendances sont des dépendances d'exécutions normales (*c.f.* définition 4.8).

DÉFINITION 4.8 (FLOT DE CONTRÔLE)
Un `flot de contrôle` d'un workflow transactionnel $Wf = (WA, \mathcal{A}xm)$ est le 2-uplet $fc_{Wf} = (AFC, \mathcal{A}xm_{fc})$ qui hérite de la définition d'un workflow transactionnel. $AFC \subseteq WA$ et $\mathcal{A}xm_{fc} \subseteq \mathcal{A}xm$ sont respectivement l'ensemble des activités et des axiomes Wf qui définissent les dépendances d'exécutions normales entre les activités de Wf. On dénote par $\mathcal{FC}ontrôle$ l'ensemble des flots de contrôle.
Soit A_0 et F les ensembles des activités de départ et de fin du workflow. $\texttt{IN}_{min}$, $\texttt{IN}_{max}$, $\texttt{OUT}_{min}$ et $\texttt{OUT}_{max}$ sont quatre **fonctions de précédence** désignant pour chaque activité a ∈ AFC un entier naturel ($\mathcal{AT} \longrightarrow$ N) comme suit :
- $\forall a \in AFC$ -A_0, $0 < \texttt{IN}_{min}(a) \leq \texttt{IN}_{max}(a) \leq \mid inDegree(a) \mid$;
- $\forall a \in AFC - F$, $0 < \texttt{OUT}_{min}(a) \leq \texttt{OUT}_{max}(a) \leq \mid outDegree(a) \mid$;
- $\forall a_0 \in A_0$ ($\texttt{IN}_{max}(a_0) = \texttt{IN}_{max}(a_0) = 0$) et $\forall f \in F$ ($\texttt{OUT}_{min}(f)$, $\texttt{OUT}_{max}(f)=0$)
- inDegree(a) est l'ensemble {d=$depNrm(b, a)$ | b ∈ AFC, d ∈ $\mathcal{D}$épendance } et outDegree(a) est l'ensemble {d=$depNrm(a, c)$ | c ∈ AFC, d ∈ $\mathcal{D}$épendance}

Le `flot de contrôle` définit une relation de précédence directe (noté '$\prec$' $\subseteq$ ((AFC - F) × (AFC -A_0)) entre les activités tel que :

$$\text{a} \prec \text{b} \stackrel{\text{def}}{=} depNrm(\text{a,b})$$
$$\text{a} \in \text{inDegree(b)}, \text{b} \in \text{outDegree(a)}$$

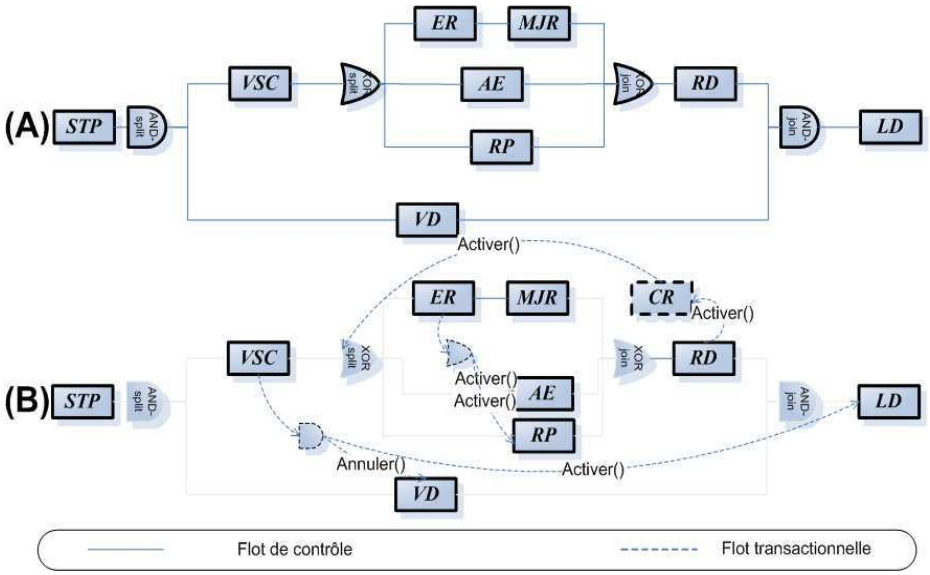
Nous définissons la relation de précédence indirecte entre deux activités (noté '$\lhd$' $\subseteq$ ((AFC - F) × ($AFC - A_0$)) comme suit :

$$\text{a} \lhd \text{b} \Leftrightarrow \exists\ \text{c} \in AFC\text{-}(A_0 \cap F) \mid ((\text{a} \prec \text{c} \wedge \text{c} \prec \text{b}) \vee (\text{a} \lhd \text{c} \wedge \text{c} \lhd \text{b}))$$

Concrètement, le `flot de contrôle` décrit des relations de précédence entre les activités, dans le cadre d'une exécution sans échec. Les fonctions $\texttt{IN}_{min}(a)$ et $\texttt{IN}_{max}(a)$ retournent le nombre d'activités minimum et maximum dont les terminaisons sont nécessaires pour l'exécution de l'activité a. Les fonctions $\texttt{OUT}_{min}$ et $\texttt{OUT}_{max}$ retournent le nombre de flots de contrôle minimum et maximum sortant suite à la terminaison (l'exécution réussie) de a, c.à.d. le nombre d'activités minimum et maximum dont l'activation dépend de a. Par exemple dans notre workflow de prêt bancaire $\texttt{IN}_{min}(LD)$=$\texttt{IN}_{max}(LD)$=2 et $\texttt{OUT}_{min}(STP) = \texttt{OUT}_{max}(STP) = 2$.

Exemple : La figure 4.4.A illustre le `flot de contrôle` du workflow de prêt bancaire défini dans la figure 2.5.

Figure 4.4 Le `flot de contrôle` et le `flot transactionnel` du workflow transactionnel de prêt bancaire



Flot transactionnel

Le `flot transactionnel` d'un WfT spécifie les mécanismes de recouvrement en cas d'échecs. Intuitivement, le `flot transactionnel` d'un WfT est défini par un ensemble de dépendances exceptionnelles. Formellement, nous définissons un `flot transactionnel` comme un WfT dont les dépendances sont seulement des dépendances exceptionnelles suite à aux échecs de ses activités.

DÉFINITION 4.9 (FLOT TRANSACTIONNEL)
Un `flot transactionnel` d'un workflow transactionnel $WfT = (WA, \mathcal{A}xm)$ est le 2-uplet $FT_{Wf} = (AFT, \mathcal{A}xm_{ft})$ qui hérite de la définition d'un workflow transactionnel tel que, $AFT \subseteq WA$ est l'ensemble des activités de Wf et $\mathcal{A}xm_{ft} \subseteq \mathcal{A}xm$ est l'ensemble des axiomes Wf désignant pour chaque échec d'une activité a ∈ AFT les axiomes d'exécutions exceptionnelles relatifs.
Plus formellement et par référence à la définition 4.7 :

$$ax \in \mathcal{A}xm_{ft} \Leftrightarrow \exists a, b \in AFT |\ a.\texttt{échouer()} \in \mathit{prédicats}(\mathcal{A}xiome(b.\texttt{act()}))$$

Concrètement, le `flot transactionnel` est l'ensemble des axiomes décrivant les dépendances exceptionnelles d'un workflow suite aux échecs d'exécution de ses activités composantes (c.à.d. l'observation de l'état `échoué()`). Ces échecs d'exécution peuvent introduire l'exécution de certaines activités qui ne figurent pas dans les activités du `flot de contrôle` comme par exemple l'activité CR dans notre workflow exemple de prêt bancaire qui est exécuté seulement à l'échec de l'activité.

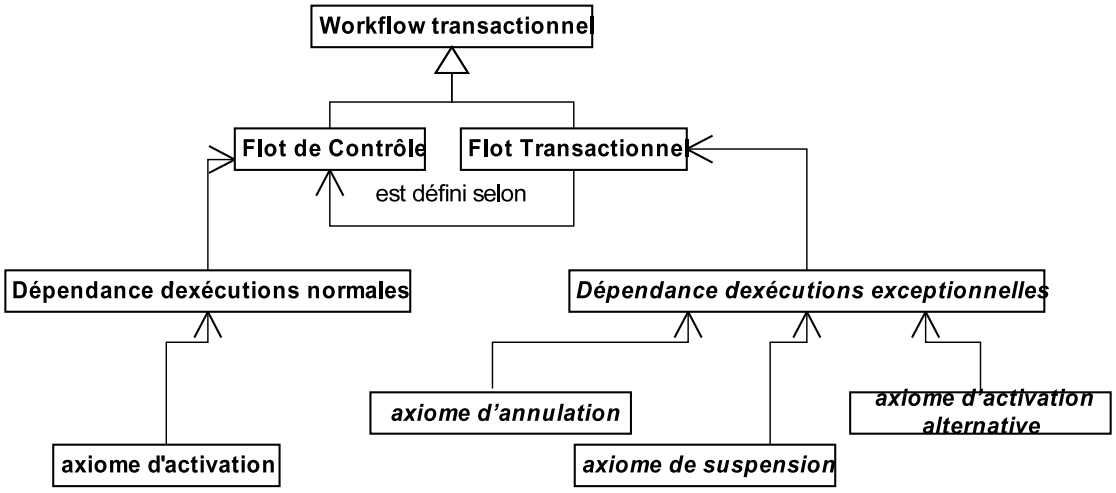
Le `flot transactionnel` est étroitement lié au `flot de contrôle` illustré par la figure 4.4.a. Ainsi les dépendances transactionnelles dépendent (de la sémantique) des dépendances d'activation. Ceci implique qu'un `flot transactionnel` est toujours défini selon un `flot de contrôle`. Dans la section 4.4.3, nous allons détailler ces relations sémantiques liant le `flot de contrôle` et le `flot transactionnel`.

Exemple La figure 4.4.b illustre le `flot transactionnel` du workflow du prêt bancaire défini à la figure 2.5.

Il est important de préciser que (*c.f.* diagramme UML de la figure 4.5) :

- Définir le `flot de contrôle` d'un WfT revient à définir pour chaque activité composante, a, ses axiomes d'activation.
- Définir le `flot transactionnel` d'un WfT revient à définir pour chaque activité composante, a, ses axiomes d'exécutions exceptionnelles incluant les axiomes d'alternative, de suspension ou d'annulation.

Figure 4.5 Dépendances et Flots dans un workflow transactionnel.



4.2.5 Synthèse

Dans cette section, nous avons présenté le concept de workflow transactionnel (WfT) en utilisant le calcul événementiel comme outil de formalisation. Un WfT contient un ensemble d'activités transactionnelles composantes caractérisées par un ensemble d'états ou d'actions spécifiant les transitions entre ces états. Nous distinguons entre les actions externes (inter-activités) et les actions internes (intra-activité) d'une activité transactionnelle. Les actions intra-activité sont activées par l'activité elle-même (l'agent responsable de l'exécuter). Les actions inter-activités sont activées par une entité externe et permettent à l'activité d'interagir avec l'extérieur. En particulier, elles permettent de mettre en œuvre l'ordonnancement des activités au sein du workflow.

Ainsi, un WfT définit l'exécution de ses activités composantes en spécifiant des axiomes sur leurs actions externes. Ces axiomes spécifient pour chaque activité composante quand elle sera activé, annulée, suspendue ou activée en tant qu'alternative. Ces axiomes définissent à un niveau d'abstraction plus haut des dépendances entre les activités composantes. Nous y distinguons les dépendances d'exécutions normales et les dépendances d'exécutions exceptionnelles. Ces dépendances expriment respectivement à leur tour à un niveau d'abstraction plus haut le `flot de`

`contrôle` et le `flot transactionnel` d'un WfT. Le `flot de contrôle` définit un ordre partiel entre les activations des activités dans une instance de workflow ou toutes les activités sont exécutées sans qu'elles échouent ou qu'elles soient annulées ou suspendues. Le `flot transactionnel` décrit les dépendances exceptionnelles suite à l'échec d'une activité composante qui spécifient les mécanismes de recouvrement. Un WfT est ainsi bien défini par son `flot de contrôle` et son `flot transactionnel`.

Dans la suite, nous montrons comment nous procédons pour spécifier le `flot de contrôle` et le `flot transactionnel` d'un WfT.

4.3 Spécification du flot de contrôle

4.3.1 Présentation

Une des raisons attribuées au manque de consensus autour des spécifications de workflow est la variété de modèles dans lesquels les procédés métiers sont spécifiés. L'absence d'une «théorie» d'organisation universelle, et de concepts de modélisation et de description de workflow standards explique et justifie finalement les différences principales dans les langages de modélisation des workflows.

Beaucoup de langages pour la conception et la spécification de workflow ont été proposés. Certains de ces langages sont basées sur des techniques de modélisation existantes telles que les réseaux de Pétri et les diagrammes d'état. Les différences dans les caractéristiques supportées par les divers systèmes de gestion de workflows commerciaux contemporains se manifestent dans les différents niveaux de pertinence et de puissance d'expressivité. Même sans qualification formelle, ces caractéristiques distinctives font référence à une sémantique fondamentalement différente (par exemple, des langages permettent des copies multiples d'une même activité en même temps et dans la même instance de workflow alors que d'autres pas, comme on peut spécifier une boucle qu'avec un point d'entrée et un point de sortie, alors que dans un autre langage on peut spécifier une boucle avec des points arbitraires d'entrée et de sortie, etc.).

Chaque langage de workflow peut être formellement décrit par un ensemble de constructeurs primitifs, de règles syntaxiques pour une composition de ces constructions. Van der Aalst et autres ont proposé un ensemble structuré de patrons [vdABtHK00] pour capitaliser et factoriser ces structures. Ces patrons fournissent la base pour une comparaison détaillée d'un certain nombre de systèmes de gestion de workflows disponibles dans le commerce. Fondamentalement, un patron permet d'exprimer d'une façon abstraite une forme récurrente dans un contexte spécifique [GHJV95]. Il est l'abstraction d'une forme concrète reproduisant les comportements récurrents dans des contextes spécifiques non arbitraires (aléatoires) et une description abstraite d'une classe d'interactions ou de dépendances. Ainsi, un patron de workflow [VTKB03] peut être vu comme une description abstraite d'une classe d'interactions récurrente basées sur les dépendances (ou primitives) d'activation.

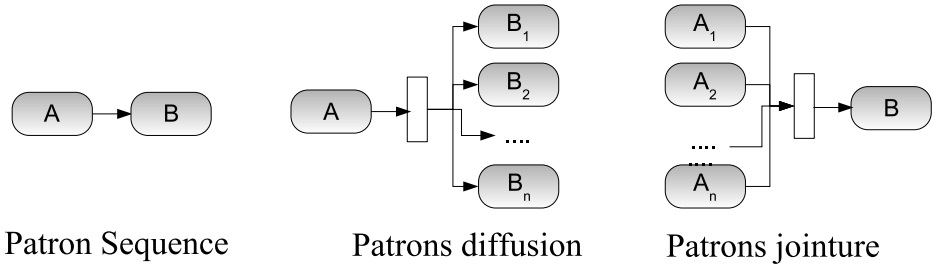
Nous avons adopté ces patrons pour décrire le `flot de contrôle` de notre modèle de workflow transactionnel. Pour ce faire, nous avons utilisé le calcul événementiel comme outil de spécification des patrons les plus communs. Ces patrons, qui capturent des aspects élémentaires du flux de contrôle du processus, mappent étroitement les définitions des concepts élémentaires du `flot de contrôle` fournis par la W*f*MC dans [Coa96].

4.3.2 Structure

Dans notre modèle, nous considérons les patrons les plus communs et les plus utilisés dans les systèmes de gestion de workflows. Nous distinguons 7 patrons : *sequence*, *diffusion parallèle*, *choix multiples*, *choix exclusif*, *synchronisation*, *OR-join*, *XOR-join* et *m-Out-Of-n* [vdABtHK00]. Dans ce qui suit, nous avons divisé les patrons de workflow en trois catégories (*c.f.* figure 4.6) : le patron séquence, les patrons de diffusion et les patrons de jointure. Ces catégories identifient les différents cheminements du flux de contrôle qui peut être unique dans la catégorie séquence ou peut être mené à se diviser dans la catégorie diffusion ou à fusionner dans la catégorie jointure. Ces cheminements décrivent quatre comportements possibles dans le `flot de contrôle` :

1. Séquentiel : Les activités sont exécutées séquentiellement (c.à.d une activité est suivie par une seule activité). Une dépendance causale exclusive lie ces deux activités.
2. Parallèle : Deux activités ou plus sont exécutées en parallèle ou en concurrence après un point de «diffusion» ou avant un point de «jointure». Nous assumons qu'aucune dépendance causale ne peut pas lier un ensemble d'activités décrivant un comportement parallèle.
3. Conditionnel : Après un point de «diffusion» ou avant un point de «jointure», une ou un ensemble d'activités est choisi pour être exécuté. Le comportement conditionnel décrit une ou plusieurs dépendances causales (non exclusives) de part et d'autre du point de «jointure» ou du point de «diffusion».
4. Boucle : Il est parfois nécessaire d'exécuter une activité ou un ensemble d'activités plusieurs fois dans le temps. Ce comportement récurrent peut aisément se formaliser par un comportement conditionnel qui décrit un flux de contrôle en boucle (un flux de contrôle conditionnel «de retour en arrière» permet cette structure de boucle).

Figure 4.6 Catégories de patrons de workflow



4.3.3 Patron *séquence*

[vdABtHK00] définit le patron *séquence* comme un comportement séquentiel exclusif liant deux activités A et B. En se référant au `flot de contrôle`, l'exécution de l'activité B dépend seulement de la terminaison de l'activité A. Et, la terminaison de A ne peut appartenir qu'au prédicat de l'axiome d'activation de B (*c.f.* définition 4.10). La figure 4.7.C illustre le `flot de contrôle` résultat de l'application du patron *séquence* à l'ensemble d'activités {ER, MJR}.

DÉFINITION 4.10 (PATRON *séquence*)
Nous définissons le patron de workflow *séquence* comme la fonction suivante :
$$\begin{array}{rccc} \textit{séquence} : & \mathcal{P}(\mathcal{AT}) & \longrightarrow & \mathcal{FC}ontrôle \\ & \{A, B\} & \longmapsto & fc_{sequence} = (AP, \mathcal{A}xm_{sequence}) \text{ tel que} \end{array}$$
- $AP = \{A, B\}$,
- Axiome d'activation de A : $\mathcal{A}xm(A.\texttt{activer()}) = \textit{axiome externe à } fc_{sequence}$
- Axiome d'activation de B : $\mathcal{A}xm(B.\texttt{activer()}) = happens(B.\texttt{activer()},t2) \leftarrow happens(A.\texttt{terminé()}, t1) \wedge t1<t2$.
- Fonction de précédence : $A \prec B \wedge \texttt{OUT}_{max}(A) = \texttt{OUT}_{min}(A) = 1 \wedge \texttt{IN}_{max}(B) = \texttt{IN}_{min}(B) = 1$

4.3.4 Patrons de «diffusion»

Cette catégorie a un opérateur de «diffusion» où un flux de contrôle se divise en plusieurs flux de contrôle qui peuvent être, selon le patron utilisé, exécutés ou pas. La dépendance entre les activités A et $\{B_i\}$ avant et après l'opérateur de «diffusion» diffère dans les trois patrons de cette catégorie : *choix exclusif*, *diffusion parallèle*, *choix multiple*.

4.3.4.1 Patron diffusion parallèle

[vdABtHK00] définit le patron *diffusion parallèle* comme un point dans le procédé d'un workflow où un flux de contrôle se divise en plusieurs flux de contrôle en parallèle, permettant ainsi une exécution parallèle (concurrente ou simultanée) d'un ensemble d'activités.

En se référant au `flot de contrôle`, un patron *diffusion parallèle* spécifie un point de «diffusion» «synchronisé» où un ensemble d'activité sera activées après la terminaison d'une autre activité (*c.f.* définition 4.11). La figure 4.7.A illustre le `flot de contrôle` résultat de l'application du patron *diffusion parallèle* à l'ensemble d'activités {STB, VSC, VD}.

DÉFINITION 4.11 (PATRON *diffusion parallèle*)
Nous définissons le patron de workflow *diffusion parallèle* par la fonction suivante :
$$\begin{array}{rccc} \textit{diffusion\_parallèle} : & \mathcal{P}(\mathcal{AT}) & \longrightarrow & \mathcal{FC}ontrôle \\ & \{A, B_1, B_2, \ldots, B_n\} & \longmapsto & fc_{ANDsplit} = (AP, \mathcal{A}xm_{ANDsplit}) \text{ tel que} \end{array}$$
- $AP = \{A, B_1, B_2, \ldots, B_n\}$,
- Axiome d'activation de A : $\mathcal{A}xm(A.\texttt{activer()}) = \textit{axiome externe à } fc_{ANDsplit}$
- Axiome d'activation de $\{B_1, B_2, \ldots, B_n\}$: $\forall i,\ 1 \leq i \leq n\ \mathcal{A}xm(B_i.\texttt{activer()}) = happens(B_i.\texttt{activer()},t2) \leftarrow happens(A.\texttt{terminé()},t1) \wedge t1<t2$.
- Fonction de précédence : $\forall i,\ 1 \leq i \leq n\ A \prec B_i \wedge \texttt{OUT}_{max}(A) = \texttt{OUT}_{min}(A) = n \wedge \texttt{IN}_{max}(B) = \texttt{IN}_{min}(B) = 1$

4.3.4.2 Patron choix exclusif

[vdABtHK00] définit le patron *choix exclusif* comme un point dans le procédé d'un workflow où, selon une condition, un seul flux de contrôle parmi d'autres sera choisi.

En se référant au `flot de contrôle`, un patron *choix exclusif* spécifie un «pseudo» point de diffusion où une activité, parmi plusieurs, sera activée suite à la terminaison d'une autre activité (*c.f.* définition 4.12). La figure 4.7.B illustre le `flot de contrôle` résultat de l'application du patron *choix exclusif* à l'ensemble d'activités {VSC, ER, AE, RP}.

DÉFINITION 4.12 (PATRON *choix exclusif*)
Nous définissons le patron de workflow *choix exclusif* par la fonction suivante :
$choix\_exclusif$: $\mathcal{P}(\mathcal{AT}) \longrightarrow \mathcal{FC}ontrôle$
$\{A, B_1, B_2, \ldots, B_n\} \longmapsto fc_{XORsplit} = (AP, \mathcal{A}xiome_{XORsplit})$ tel que
- $AP = \{A, B_1, B_2, \ldots, B_n\}$,
- Axiome d'activation de A : $\mathcal{A}xm(A.\texttt{activer()}) = axiome\ externe\ à\ fc_{XORsplit}$
- Axiome d'activation de $\{B_1, B_2, \ldots, B_n\}$: $\forall i,\ 1 \leq i \leq n\ \mathcal{A}xiome(B_i.\texttt{activer()}) = happens(B_i.\texttt{activer()},t2) \leftarrow happens(A.\texttt{terminé},\ t1) \wedge t1<t2\ wedge\ happens(c_i,t)$ | il y a toujours une seule condition c_j $(1 \leq j \leq n)$ évaluée à vrai après la terminaison de A.
- Fonction de précédence : $\forall i,\ 1 \leq i \leq n\ A \prec B_i \wedge \texttt{OUT}_{max}(A) = \texttt{OUT}_{min}(A) = 1 \wedge \texttt{IN}_{max}(B_i) = \texttt{IN}_{min}(B_i) = 1$

4.3.4.3 Patron choix multiples

[vdABtHK00] définit le patron *choix multiples* comme un point dans le procédé d'un workflow où, selon une condition, un sous ensemble d'activités sera choisi.

En se référant au `flot de contrôle`, un patron *choix multiples* spécifie un point de «diffusion» où un sous ensemble d'activités sera choisi pour être activé suite à la terminaison d'une autre activité (*c.f.* définition 4.13).

DÉFINITION 4.13 (PATRON *choix multiples*)
Nous définissons le patron de workflow *choix multiples* par la fonction suivante :
`choix_multiples` : $\mathcal{P}(\mathcal{AT}) \longrightarrow \mathcal{FC}ontrôle$
$\{A, B_1, B_2, \ldots, B_n\} \longmapsto fc_{XORsplit} = (AP, \mathcal{A}xiome_{ORsplit})$ tel que
- $AP = \{A, B_1, B_2, \ldots, B_n\}$,
- Axiome d'activation de A : $\mathcal{A}xm(A.\texttt{activer()}) = axiome\ externe\ à\ fc_{ORsplit}$
- Axiome d'activation de $\{B_1, B_2, \ldots, B_n\}$: $\forall i,\ 1 \leq i \leq n\ \mathcal{A}xm(B_i.\texttt{activer()}) = happens(B_i.\texttt{activer()},t2) \leftarrow happens(A.\texttt{terminé},\ t1) \wedge t1<t2 \bigwedge happens(c_i,t)$ | il y a plusieurs conditions c_j $(1 \leq j \leq n)$ évaluées à vrai après la terminaison de A.
- Fonction de précédence : $\forall i,\ 1 \leq i \leq n\ A \prec B_i \wedge \texttt{OUT}_{max}(A) = n, \texttt{OUT}_{min}(A) = 2 \wedge \texttt{IN}_{max}(B_i) = \texttt{IN}_{min}(B_i) = 1$

4.3.5 Patrons de «jointure»

Cette catégorie a un opérateur de «jointure» où plusieurs flux de contrôle fusionnent dans un unique flux de contrôle. Le nombre de branches nécessaires pour l'activation de l'activité B après un opérateur de «jointure» dépend du patron utilisé. La dépendance entre les activités A_i et B avant et après l'opérateur de «jointure» diffère dans les trois patrons de cette catégorie : les patrons : *synchronisation*, *jointure simple*, et *M-out-of-N*.

4.3.5.1 Patron synchronisation

[vdABtHK00] définit le patron *synchronisation* comme un point dans le procédé d'un workflow où plusieurs flux de contrôle en parallèle convergent et se synchronisent en un seul flux de contrôle.

En se référant au `flot de contrôle`, un patron *synchronisation* spécifie un point de «jointure» «synchronisé» où une activité sera activée après la terminaison d'un ensemble d'autres activités

(*c.f.* définition 4.14). La figure 4.7.E illustre le `flot de contrôle` résultat de l'application du patron *synchronisation* à l'ensemble d'activités {RD, VD, LD}.

DÉFINITION 4.14 (PATRON *synchronisation*)
Nous définissons le patron de workflow *synchronisation* par la fonction suivante :
$\texttt{synchronisation} : \mathcal{P}(\mathcal{AT}) \longrightarrow \mathcal{FC}ontrôle$
$\{A_1, A_2, \ldots, A_n, B\} \longmapsto fc_{ANDjoin} = (AP, \mathcal{A}xiome_{ANDjoin})$ tel que
- $AP = \{A_1, A_2, \ldots, A_n, B\}$,
- Axiome d'activation de B : $\mathcal{A}xm(B.\texttt{activer()}) = happens(B.\texttt{activer()},t2) \leftarrow \bigwedge_{i=1\ldots n} happens(A_i.\texttt{terminer()},t1) \wedge t1<t2$
- Axiome d'activation de $\{A_1, A_2, \ldots, A_n\} : \forall i, 1 \leq i \leq n\ \mathcal{A}xm(A_i.\texttt{activer()}) = axiome\ externe\ à\ fc_{ANDjoin}$.
- Fonction de précédence : $\forall i, 1 \leq i \leq n\ A_i \prec B \wedge \texttt{OUT}_{max}(A_i) = \texttt{OUT}_{min}(A) = 1 \wedge \texttt{IN}_{max}(B) = \texttt{IN}_{min}(B) = n$

4.3.5.2 Patron jointure simple

[vdABtHK00] définit le patron *jointure simple* comme un point dans le procédé d'un workflow où un flux de contrôle d'un ensemble de flux de contrôle en parallèle converge dans un «pseudo» point de «jointure».

En se référant au `flot de contrôle`, un patron *jointure simple* spécifie un «pseudo» point de «jointure» où une activité sera activée suite à la terminaison d'une seule activité parmi plusieurs appartenant à des flux de contrôle parallèles (*c.f.* définition 4.15). La figure 4.7.D illustre le `flot de contrôle` résultat de l'application du patron *jointure simple*à l'ensemble d'activités {MJR, AE, RP, RD}.

DÉFINITION 4.15 (PATRON *jointure simple*)
Nous définissons le patron de workflow *jointure simple* par la fonction suivante :
$\texttt{jointure\_simple} : \mathcal{P}(\mathcal{AT}) \longrightarrow \mathcal{FC}ontrôle$
$\{A_1, A_2, \ldots, A_n, B\} \longmapsto fc_{XORjoin} = (AP, \mathcal{A}xiome_{XORjoin})$ tel que
- $AP = \{A_0, A_1, \ldots, A_n, B\}$,
- Axiome d'activation de B : $\mathcal{A}xm(B.\texttt{activer()}) = happens(B.\texttt{activer()},t2) \leftarrow \bigvee_{i=1\ldots n} happens(A_i.\texttt{terminer()},t1) \wedge t1<t2$
- Axiome d'activation de $\{A_1, A_2, \ldots, A_n\} : \forall i, 1 \leq i \leq n\ \mathcal{A}xm(A_i.\texttt{activer()}) = axiome\ externe\ à\ fc_{XORjoin}$.
- Fonction de précédence : $\forall i, 1 \leq i \leq n\ A_i \prec B \wedge \texttt{OUT}_{max}(A_i) = 0, \texttt{OUT}_{min}(A) = 1 \wedge \texttt{IN}_{max}(B) = \texttt{IN}_{min}(B) = 1$

4.3.5.3 Patron M-out-of-N

[vdABtHK00] définit un patron *M-out-of-N* comme un point dans le procédé d'un workflow où m flux de contrôle parmi n s'exécutant en parallèle convergent et se synchronisent en un seul flux de contrôle.

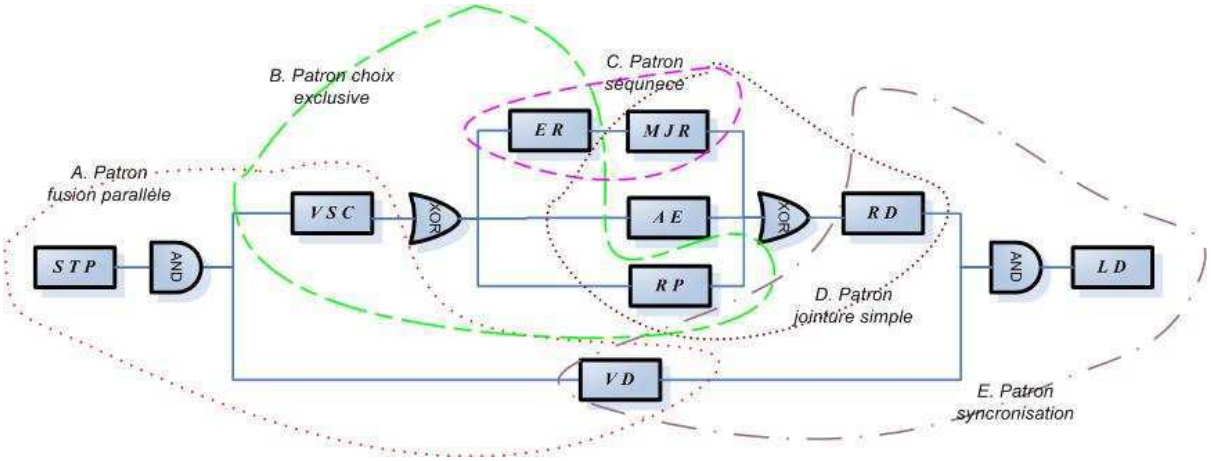
En se référant au `flot de contrôle`, un patron *M-out-of-N* spécifie un point de «jointure» où une activité sera activée après la terminaison de m activités parmi n s'exécutant en parallèle (*c.f.* définition 4.16).

DÉFINITION 4.16 (PATRON *M-out-of-N*)
Nous définissons le patron de workflow *M-out-of-N* par la fonction suivante :

$$\texttt{M-out-of-N} : \quad \mathcal{P}(\mathcal{AT}) \longrightarrow \mathcal{FC}ontrôle$$
$$\{A_1, A_2, \ldots, A_n, B\} \longmapsto fc_{XORjoin} = (AP, \mathcal{A}xiome_{ORjoin}) \text{ tel que}$$

- $AP = \{A_0, A_1, \ldots, A_n, B\}$,
- Axiome d'activation de B : $\mathcal{A}xm(B.\texttt{activer()}) = happens(B.\texttt{activer()},t2) \leftarrow \bigwedge_{i=1\ldots m} happens(A_{i,j}.\texttt{terminer()},t1) \wedge t1<t2$; $A_{i,j}$ appartient à la j^{ieme} partition de cardinal m de $\{A_0, A_1, \ldots, A_n\}$
- Axiome d'activation de $\{A_1, A_2, \ldots, A_n\} : \forall i, 1 \leq i \leq n\ \mathcal{A}xm(A_i.\texttt{activer()}) = axiome\ externe\ à\ fc_{ORjoin}$.
- Fonction de précédence : $\forall i, 1 \leq i \leq n\ A_i \prec B \wedge \texttt{OUT}_{max}(A_i) = 0, \texttt{OUT}_{min}(A_i) = 1 \wedge \texttt{IN}_{max}(B) = m, \texttt{IN}_{min}(B) = n$

Figure 4.7 Le `flot de contrôle` du workflow du prêt bancaire peut être défini comme une union cohérente de patrons de workflow



4.3.6 Synthèse

Dans cette section, nous avons introduit le concept de patron de workflow que nous utilisons pour spécifier le `flot de contrôle` d'un WfT. Nous définissons le `flot de contrôle` d'un WfT comme une union cohérente de patrons. Nous avons utilisé le calcul événementiel comme formalisme pour capturer les propriétés comportementales de base de chacun des patrons de workflow étudiés. Ces propriétés comportementales décrivent les cheminements du flux de contrôle dans chaque patron indépendamment du contexte d'implantation du patron ou du contexte métier, laissant ainsi **une flexibilité** pour la spécification des besoins métiers pour les concepteurs. Par ailleurs, ces propriétés comportementales nous seront utiles, par la suite, dans nos techniques de découverte (*c.f.* chapitre suivant) pour découvrir le `flot de contrôle` à partir des traces d'exécutions.

4.4 Spécification du flot transactionnel

Un workflow transactionnel (WfT) exploite le comportement transactionnel de ses activités composantes pour spécifier des mécanismes de recouvrement. Dans cette section, nous étudions les spécifications transactionnelles d'un workflow en modélisant son `flot transactionnel`. Nous introduisons ainsi un modèle transactionnel qui va étendre le `flot de contrôle` en ajoutant des états intermédiaires décrivant le `flot transactionnel` en cas d'échecs d'exécution.

Dans notre modèle, nous distinguons entre les flux transactionnels inter et intra-activité. En effet, l'introduction de la notion de flux transactionnel intra-activité (*c.f.* section 4.4.1) nous permet de décrire les propriétés transactionnelles des activités transactionnelles que nous considérons. Nous montrons, en particulier, comment nous modélisons le comportement interne d'une activité selon ses propriétés transactionnelles. La section 4.4.2 dévoile le côté transactionnel des dépendances inter-activités suite à un échec d'exécution, que nous exploitons pour définir les différents mécanismes de recouvrement à travers les dépendances d'activation alternative, d'annulation et de suspension. Nous montrons finalement, dans la section 4.4.3, comment le `flot transactionnel` est lié sémantiquement au `flot de contrôle`.

4.4.1 Flux transactionnel intra-activité

Une activité transactionnelle est une activité dont le comportement interne manifeste des propriétés transactionnelles. Dans le paragraphe 4.4.1.1, nous décrivons les propriétés transactionnelle que nous considérons et sont inspirées par les travaux menés dans [MRKS92]. À chacune de ces propriétés est associée un diagramme à transitions d'états qui modélise son comportement transactionnel (*c.f.* paragraphe 4.4.1.2). Ce diagramme décrit les états possibles par lesquels une instance d'une activité peut passer durant son cycle de vie et les transitions possibles entre ces états. De l'ensemble d'états et des transitions dépendent les propriétés transactionnelles de l'activité.

4.4.1.1 Propriétés transactionnelles d'une activité

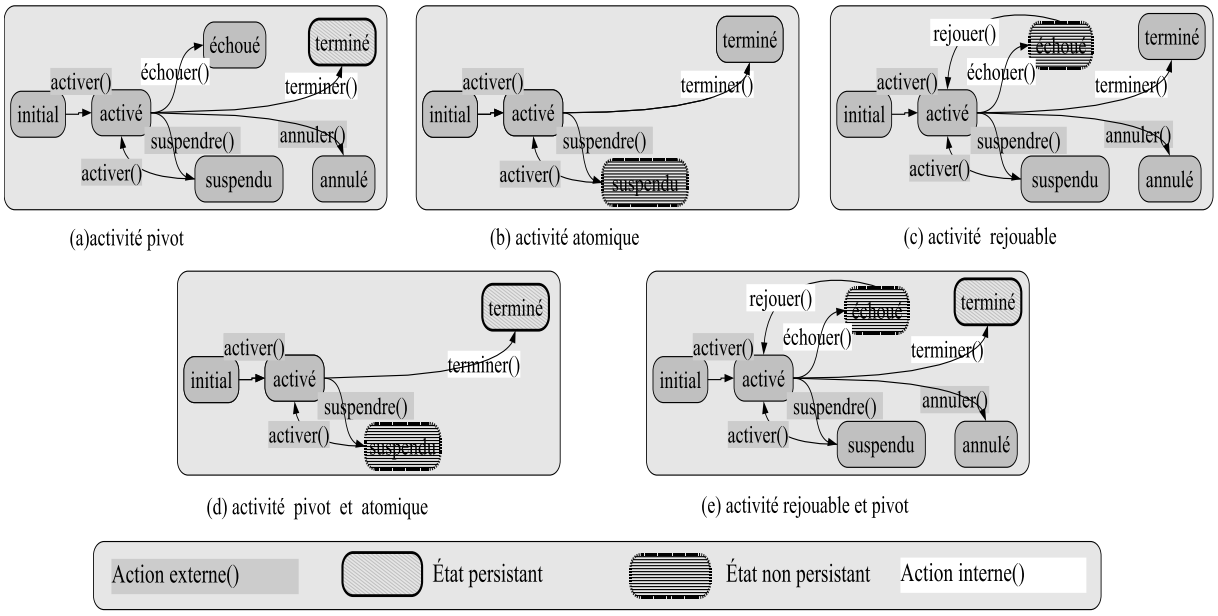
Une activité transactionnel (AT) est une activité dont le comportement interne manifeste des propriétés transactionnelles. Les propriétés transactionnelles que nous considérons dans notre approche sont **rejouable**, **atomique** et **pivot**. Une activité a est dite rejouable (a^r) si elle se termine toujours avec succès après un nombre fini d'activations. Une activité a est dite atomique (a^a) si elle ne peut pas être ni annulée ni suspendue définitivement et qu'elle est sûre de terminer avec succès après son activation. Une activité a est dite pivot (a^p) si une fois qu'elle se termine avec succès, elle ne peut pas être re-exécutée, et ainsi les effets de son exécution sont persistants. L'ensemble des combinaisons de propriétés possibles est $\{\emptyset; r; a; p; (p, a); (r, p)\}$.

Les propriétés que nous utilisons ont été introduites dans certains modèles transactionnels avancés (principalement le modèle des transactions flexibles). L'utilisation et l'appellation (transactionnelle) de ces propriétés sont justifiées par le fait qu'elles permettent de caractériser le comportement d'une activité dans un contexte d'échec d'exécution. Ainsi, la propriété **rejouable** permet d'informer si une activité est susceptible d'échouer ou non. De même la propriété **atomique** informe si une activité peut être interrompue ou annulée au cours d'exécution, alors que **pivot** informe si les effets de cette activité peuvent être sémantiquement défaits ou non.

4.4.1.2 Diagramme à transitions d'états d'une activité transactionnelle

Comme nous l'avons indiqué dans la section 4.2.2, les ensembles des états et des actions peuvent être étendus ou restreints pour exprimer les propriétés transactionnelles des activités. Nous avons besoin d'ajouter des axiomes à chacune de ces propriétés transactionnelles, en plus des axiomes du domaine de définition décrits dans la section 4.2.2, pour décrire les nouvelles caractéristiques transactionnelles qu'elles spécifient.

Figure 4.8 Diagramme à transitions d'états et propriétés transactionnelles d'une activité transactionnelle



Activité pivot : Le diagramme à transitions d'états d'une activité pivot est illustré par la figure 4.8.a. Il montre que l'état `terminé` est un état persistant. En effet, une activité est pivot si elle atteint l'état `terminé` elle ne peut pas être défaite par une re-activation ou une annulation. Formellement, une activité pivot, at^p, est une activité transactionnelle qui affine le **domaine de définition** initial comme suit :

L'ensemble minimal d'états : `initial`, `activé`, `annulé`, `suspendu`, `échoué` et `terminé`.

L'ensemble minimal d'actions : `activer()`, `annuler()`, `suspendre()`, `échouer()` et `terminer()`).

Axiome(s) additionnelle(s) : $\nexists(a() \in Inter(at^p) \cup Intra(at^p)) \mid (terminates(at^p.\text{a}()$,`terminé`(at^p), t)$\leftarrow HoldsAt($`terminé`(at^p),t) $\wedge\ Happens(at^p.a()$,t)).

Activité atomique : Le diagramme à transitions d'états d'une activité atomique est illustré par la figure 4.8.a. Il montre que cette activité est sûre de se terminer avec succès une fois qu'elle est activée et qu'elle ne peut pas échouer. Ainsi, l'état `suspendu` dans une activité atomique ne doit être un état final de l'activité, c'est pourquoi il est défini comme non persistant. Formellement, une activité atomique, at^a, est une activité transactionnelle qui affine le **domaine de définition** initial comme suit :

L'ensemble minimal d'états : `initial`, `activé`, `suspendu` et `terminé`.

L'ensemble minimal d'actions : `activer()`, `suspendre()`, `terminer()` et `rejouer()`.

Axiome(s) additionnelle(s) : $initiates(at^a.\texttt{activer()}, \texttt{activé}(at^a), t2) \wedge terminates(at^a.\texttt{activer()}, \texttt{suspendu}(at^a), t2) \leftarrow Happens(at^a.\texttt{activer()}, t1) \leftarrow HoldsAt(\texttt{suspendu}(at^a), t2) ; t1{>}t2$.

Activité rejouable : [21] une activité rejouable possède une action `rejouer()` qui ramène l'activité à son état `activé` à chaque fois qu'elle échoue. En conséquence l'état `échoué` est un état non persistent et ne peut pas être un état final pour l'activité. La figure 4.8.c illustre le diagramme à transitions d'états d'une activité rejouable. Formellement, une activité rejouable, at^r, est une activité transactionnelle qui affine le **domaine de définition** initial comme suit :

L'ensemble minimal d'états : `initial, suspendu, activé, échoué, annulé, terminé`.

L'ensemble minimal d'actions : `activer(), annuler(),suspendre, échouer(), terminer(), rejouer()`.

Axiome(s) additionnelle(s) : $Happens(at^r.\texttt{rejouer()}, t) \leftarrow HoldsAt(\texttt{échoué}(at^r), t)$.

Activité pivot et atomique : une activité peut être à la fois pivot et atomique. Le diagramme à transitions d'états d'une activité pivot et atomique qui est illustré dans la figure 4.8.d montre qu'on a gardé les caractéristiques d'une activité pivot tout en supprimant les états `échoué` et `annulé` qui ne font pas partie des caractéristiques transactionnelles d'une activité atomique et en précisant que l'état `échoué` est non persistant. Formellement, une activité pivot et atomique, $at^{p,a}$, est une activité transactionnelle qui affine le **domaine de définition** :

L'ensemble minimal d'états : `initial`, `activé`, `suspendu` et `terminé`.

L'ensemble minimal d'actions `activer()`, `suspendre()` `échouer()` et `terminer()`), `rejouer`.

Axiome(s) additionnelle(s) : $initiates(at^{p,a}.\texttt{activer()}, \texttt{activé}(at^{p,a}), t2) \wedge terminates(at^{p,a}.\texttt{activer()}, \texttt{suspendu}(at^{p,a}), t2) \leftarrow Happens(at^{p,a}.\texttt{activer()}, t1) \leftarrow HoldsAt(\texttt{suspendu}(at^{p,a}), t2) ; t1{>}t2$.

$\nexists(a() \in Intra(at^{p,a}) \cup Inter(at^p), (terminates(at^{p,a}.a(), \texttt{terminé}(at^{p,a}), t) \leftarrow HoldsAt(\texttt{terminé}(at^{p,a}), t) \wedge Happens(at^{p,a}.a(), t))$

4.4.2 Flux transactionnel inter-activités

4.4.2.1 Dépendances transactionnelles inter-activités

Après un échec d'exécution d'une activité (c.à.d. l'exécution de l'action `échouer()`), le workflow se trouve dans un état incohérent. Des actions inter-activités, qui spécifient des dépendances d'exécutions exceptionnelles (*c.f.* définition 4.7), sont alors mises à l'œuvre par les entités externes (programmateur, intervention humaine, etc.) pour permettre à l'activité échouée d'agir avec l'extérieur pour recouvrir un état cohérent. Le but est de remettre le workflow échoué de nouveau dans un état sémantiquement acceptable. Ainsi, l'état incohérent peut alors être traité

[21] Une activité rejouable et pivot possède les mêmes transitions et états qu'une activité rejouable. Cependant dans une activité rejouable et pivot, il est précisé que l'état `terminé` est un état persistent (c.à.d l'activité est pivot).

et l'exécution reprise, grâce aux mécanismes de recouvrement qui la ramèneront à un nouveau point cohérent avec l'espoir qu'elle se terminera alors avec succès.

Dans la suite, nous allons décrire les différents types de ces dépendances. Ces dépendances que nous qualifions comme transactionnelles héritent des dépendances d'exécutions exceptionnelles mais décrivent le comportement et les interactions avec les autres activités suite à un échec (c.à.d. l'exécution de l'action `échouer()`). Ces dépendances transactionnelles permettent de décrire les mécanismes de recouvrement nécessaires à la poursuite de l'exécution de l'instance de workflow.

4.4.2.2 Dépendance d'alternative

Les dépendances d'alternative permettent de définir des alternatives d'exécution comme mécanismes de recouvrement. Ainsi, une dépendance d'alternative de a_1 vers a_2 existe ***ssi*** l'échec de a_1 peut déclencher l'activation de a_2.

Définition 4.17 (Dépendance transactionnelle d'alternative)
Une dépendance transactionnelle d'alternative de a_1 vers s_2, notée $depAlt(a_1, a_2)$, existe ***ssi*** une dépendance d'exécutions exceptionnelles existe entre une action source (a_1.`échouer()`) et une action destination (a_2.`activer()`). En termes d'états et d'axiomes, $depAlt(a_1, a_2)$ existe ***ssi*** le prédicat de déclenchement de l'action `échouer()` de a_1 est un prédicat de l'axiome de l'action `activer()` de a_2.
Plus formellement et par référence à la définition 4.7 :

$$depAlt(a_1, a_2) \stackrel{\text{def}}{=} depEx(a_1.\texttt{échouer()}, a_2.\texttt{activer()}).$$

$$\exists\ depAlt(a_1, a_2) \Longleftrightarrow \exists happens(a_1.\text{échouer()},\text{t}) \in prédicats(\mathcal{A}xm(a_2.\texttt{activer}))$$

Nous distinguons deux types d'alternative : une alternative en avant si l'activité a_2 n'est pas exécutée avant l'échec de a_1 et une alternative en arrière si l'activité a_2 est exécutée avant l'échec de a_1.

Exemple : Le workflow de prêt bancaire spécifie pour l'activité ER des dépendances d'alternative en avant vers l'activité AE ($depAlt(ER, AE)$) ou exclusivement vers l'activité RP ($depAlt(ER, RP)$). Ainsi, l'activité AE (ou l'activité RP) sera activée quand ER échoue.

Notons bien que l'axiome d'activation de l'action `activer()` d'une activité a, a.`activer()`, peut être défini par le `flot de contrôle` à travers une dépendance d'exécutions normales ou par le `flot transactionnel` à travers une dépendance d'alternative.

4.4.2.3 Dépendance transactionnelle d'annulation

Une dépendance d'annulation permet de signaler les échecs d'exécution d'une activité à d'autre(s) activité(s) s'exécutant en parallèle en provoquant leur annulation et l'arrêt définitif de leur(s) exécution(s). Ainsi, une dépendance d'annulation de a_1 vers a_2 existe ***ssi*** l'échec de $a1$ peut déclencher l'annulation de a_2.

DÉFINITION 4.18 (DÉPENDANCE TRANSACTIONNELLE D'ANNULATION)
Une dépendance d'annulation de a_1 vers a_2, notée $depAnl(a_1, sa_2)$, existe ***ssi*** une dépendance d'exécutions exceptionnelles existe entre une action source (a_1.`échouer()`) et une action destination (a_2.`annuler()`). En termes d'états et d'axiomes, $depAnl(a_1, a_2)$ existe ***ssi*** les prédicats de déclenchement de l'action `échouer()` de a_1 est un prédicat de l'axiome de l'action `annuler()` de a_2.
Plus formellement et par référence à la définition 4.7 :

$$depAnl(a_1, a_2) \stackrel{\text{def}}{=} depEx(a_1.\texttt{échouer()}, a_2.\texttt{annuler()})$$

$$\exists\ depAnl(a_1, a_2) \Longleftrightarrow \exists\ \text{happens}(a_1.\texttt{échouer()},\text{t}) \in \textit{prédicats}(\mathcal{A}xm(a_2.\texttt{annuler}))$$

Exemple Le workflow de prêt bancaire spécifie une dépendance d'annulation de l'activité VSC vers l'activité VD ($depAnl(VSC, VD)$). Ainsi, l'activité VD sera `annulée` si VSC échoue.

4.4.2.4 Dépendance transactionnelle de suspension

Une dépendance transactionnelle de suspension permet de signaler les échecs d'exécution d'une activité à d'autre(s) activité(s) s'exécutant en parallèle en provoquant leur suspension et l'arrêt temporaire de leurs exécutions. Cet arrêt temporaire suppose que l'activité suspendue peut être réactivée par la suite après l'exécution du mécanisme de recouvrement. Ainsi, une dépendance de suspension de a_1 vers a_2 existe ***ssi*** l'échec, de a_1 peut déclencher la suspension de a_2.

DÉFINITION 4.19 (DÉPENDANCE TRANSACTIONNELLE DE SUSPENSION)
Une dépendance de suspension de a_1 vers a_2, notée $depSus(s_1, s_2)$, existe ***ssi*** une dépendance d'exécutions exceptionnelles existe entre une action source (a_1.`échouer()`) et une action destination (a_2.`suspendre()`). En termes d'états et d'axiomes, $depSus(a_1, a_2)$ existe ***ssi*** les prédicats de déclenchement de l'action `échouer()` de a_1 est un prédicat de l'axiome de l'action `suspendre()` de a_2.
Plus formellement et par référence à la définition 4.7 :

$$depSus(a_1, a_2) \stackrel{\text{def}}{=} depEx(a_1.\texttt{échouer()}, a_2.\texttt{suspendre()})$$

$$\exists\ depSus(a_1, a_2) \Longleftrightarrow \exists\ \text{happens}(a_1.\texttt{échouer()},\text{t}) \in \textit{prédicats}(\mathcal{A}xm(a_2.\texttt{suspendre()})$$

4.4.3 Mécanismes de recouvrement dans les workflows transactionnels

4.4.3.1 Mécanismes de recouvrement

Un workflow transactionnel (WfT) exploite les dépendances transactionnelles de ses activités composantes pour spécifier des mécanismes de recouvrement. Concrètement, le workflow transactionnel (WfT) doit décider, après un échec d'activité, si un état incohérent est atteint. Selon cette décision soit une procédure de reprise complexe doit être commencée, soit l'exécution de l'instance va continuer comme elle est. Le défi principal de ce fait et d'identifier est d'atteindre un état cohérent à partir duquel le workflow peut continuer l'exécution.

Un point d'état cohérent est une étape d'exécution (équivalent au dernier point de validation[22] dans des transactions de bases de données) qui représente un état intermédiaire acceptable d'exécution par rapport à une perspective métier. Ce point cohérent dans le flux de contrôle est un point de décision où certaines mesures peuvent être prises pour résoudre un problème d'échecs d'exécutions [DDS97]. Contrairement aux systèmes de bases de données, le point cohérent est choisi en fonction de la sémantique d'exécution du procédé métier et ne dépend pas uniquement de la transaction où l'activité échoue. En effet, un point cohérent de reprise d'exécution est défini par rapport à l'échec d'une activité et sera par la suite dépendant de cette activité. Par exemple, le point cohérent, suite à l'échec de l'activité ER dans notre exemple de motivation, est situé avant les activités AE et RP. Quant au point cohérent relatif à l'activité LD il s'agit de l'activité elle-même puisqu'elle est rejouable. Ce choix dépend ainsi de la sémantique et du contexte métier choisi par le concepteur lors de la conception de son procédé métier.

La procédure de recouvrement, si le point cohérent n'est pas l'activité elle-même, est initialisée par une dépendance alternative (*c.f.* définition 4.17). Selon la localisation du point cohérent, nous avons identifié deux types de dépendances alternatives : alternative en arrière et alternative en avant. En plus, ramener le workflow à un état sémantiquement acceptable peut également nécessiter de compenser les activités déjà réalisées par des «nouvelles» activités de compensation qui défont sémantiquement l'activité échouée [KMO98b]. Ceci peut induire l'exécution de nouvelles activités qui ne se trouvent pas dans le `flot de contrôle` et qui sont exécutées seulement en cas d'échec d'exécution afin de compenser ses effets. Par exemple le workflow de prêt bancaire montre que suite à l'échec de l'activité RD, l'instance de workflow exécute une nouvelle activité CR qui n'existe pas dans le flux de contrôle, afin de compenser les effets de cet échec et reprendre le processus de décision à partir du point cohérent se trouvant après l'activité VSC.

En outre, un échec d'activité peut causer une fin non régulière et anormale (annulation ou suspension) d'une ou plusieurs activités actives. Si une telle situation se produit, l'échec d'une activité induit l'annulation d'autres activités. Ce comportement est décrit par la dépendance d'annulation ou de suspension (*c.f.* définitions 4.18 et 4.19). Le workflow de notre exemple de motivation indique que suite à l'échec de l'activité VSC, l'activité VD est annulée si elle n'a pas encore fini son exécution, puisque cet échec de VSC cause l'arrêt final de l'instance du workflow.

En récapitulant, après un échec d'une activité, nous avons, pour recouvrir un état cohérent, les possibilités suivantes :

- L'échec de l'activité n'a pas d'incidence sur l'exécution de l'instance de workflow. Ainsi, on peut continuer sans aucun mécanisme spécifique de recouvrement (c.à.d continuer l'exécution selon le `flot de contrôle`).
- L'échec de l'activité nécessite qu'un mécanisme de recouvrement soit activé. Ainsi, l'activité doit être recouvrable (*c.f.* définition 4.20). Nous distinguons trois mécanismes de recouvrement différents :
  - L'activité est **rejouable** : Un re-exécution automatique jusqu'au succès est adoptée. Un tel mécanisme est généralement utilisé si l'échec de l'activité est sans conséquence. En d'autres termes, l'effet général de son exécution ou non avec succès n'influe pas sur l'exécution d'autres activités mais interrompt nécessairement l'exécution de l'instance ce qui nécessite une re-exécution jusqu'au succès. Ainsi, ce type d'activité, que l'on peut aussi appeler activité neutre ou idempotente, peut être exécuté une ou plusieurs fois sans avoir de conséquences, ce qui offre un dispositif de gestion d'erreurs très confortable dans l'exécution de workflows.
  - Le recouvrement **en arrière** : si l'activité n'est pas neutre. En d'autres termes, son échec

[22] exécution de la transaction `commit`

a une influence sur d'autres activités alors une procédure de recouvrement est nécessaire afin d'atteindre le précédent point d'exécution cohérent. Sans cette procédure de recouvrement, l'exécution de l'instance de workflow reste interrompue causant l'échec global de l'exécution de l'instance du workflow. Par ailleurs, la procédure de recouvrement en arrière peut nécessiter l'exécution d'une activité de compensation qui «défait» sémantiquement et enlève des effets secondaires incohérents dus à l'échec. Ce retour à un état antérieur est initialisé par une dépendance alternative en arrière.

– Le recouvrement **en avant** : Le concepteur peut choisir après l'échec d'une activité de changer le traitement initial de l'instance en choisissant un chemin alternative et en continuant l'exécution de l'instance de workflow vers l'avant. Concrètement, l'instance de workflow exécute une autre activité pas encore exécutée et essaye de terminer correctement l'exécution de workflows. Ce mécanisme de recouvrement est initialisé par une dépendance alternative en avant.

DÉFINITION 4.20 (ACTIVITÉ RECOUVERABLE)
a est recouverable (a^c) *ssi* elle propose un mécanisme de recouvrement (propriété rejouable ou/et dépendances alternatives d'exécution, de suspension ou d'annulation.)

4.4.3.2 Relations sémantiques dans les mécanismes de recouvrement

La spécification des mécanismes de recouvrement définie à travers le comportement transactionnel doit respecter quelques «règles» sémantiques partiellement dépendantes du flux de contrôle. En effet, le comportement transactionnel représenté par les dépendances de suspension, d'annulation et d'alternative et les propriétés transactionnelles sont étroitement liées aux dépendances d'exécutions normales décrites à travers les patrons de workflow. Par exemple, une dépendance de suspension ou d'annulation de a_1 vers a_2 ne peut exister que si les deux activités ne dépendent pas l'une de l'autre au sens du flux de contrôle normal et sont en concurrence. Concrètement, ces relations sémantiques sont décrites dans la règle 4.1.

La première et la deuxième règle expriment le fait qu'une dépendance d'annulation ou de suspension ne peut se produire qu'entre des activités concurrentes. La troisième règle indique qu'une activité atomique ne peut pas faire partie d'une dépendance de suspension même si elle est exécutée en concurrence avec d'autres activités qui peuvent échouer. Alors que la quatrième règle spécifie les conditions sous lesquelles une dépendance d'alternative (recouvrement en arrière ou en avant) peut être invoquée. La cinquième règle spécifie les activités vitales à l'exécution d'une instance de workflow. La sixième règle exprime qu'un point cohérent ne peut se trouver dans un flux parallèle. la dernière règle est une conséquence du caractère persistant d'une activité pivot.

Ces relations sont communes à tous les workflow transactionnels indépendamment du procédé métier sujet. Elles s'appliquent parfaitement sur notre exemple de motivation. En effet, on spécifie un mécanisme de recouvrement pour toutes les activités qui échouent respectant les contraintes évoquées dans ces relations.

Par ailleurs, ces relations peuvent être enrichies par le concepteur qui peut définir d'autres règles qui sont dépendantes de la sémantique propre de son procédé métier. En plus, notre modèle transactionnel peut être étendu. En effet, d'autres types de propriétés ou de dépendances transactionnelles peuvent être ajoutées à notre modèle de transactions qui va en conséquence restructurer ses relations sémantiques pour intégrer ces modifications. Les travaux de Bhiri et autres [BGP04b; BGP04a; BPG05] proposent un modèle transactionnel flexible qu'on peut facilement utiliser pour

Règle 4.1 (Relations sémantiques)
Pour les patrons de workflow :

1. **R1** : `choix exclusif`, `jointure simple` et `séquence` :

   Il n'y a aucune dépendance d'annulation entre les activités non atomiques de ces patrons ;

2. **R2** : Pour les patrons `choix exclusif`, `jointure simple` et `séquence` :

   Il n'y a aucune dépendance de suspension entre les activités de ces patrons ;

3. **R3** : Pour les activité des flux parallèles des patrons `synchronisation`, `M-out-of-N`, `diffusion parallèle`, `choix multiples` :

   Il n'y a pas de dépendances d'annulation vers les activités atomiques ;

4. **R4** : Pour les patrons `synchronisation` `M-out-of-N` :

   Le point cohérent d'une activité recouvrable et non rejouable doit être hors des flux de contrôles parallèles ;

5. **R5** : De tous ces patrons sauf les activités des flux parallèles des combinaison des patrons (`diffusion parallèle` et (`M-out-of-N` ou `jointure simple`)) et des (`choix multiples` et `jointure simple` :

   Toutes les activités doivent être recouvrables en cas d'échec ou doivent être atomiques ;

6. **R6** :Pour les patrons `synchronisation`, `M-out-of-N` :

   Il ne peut y avoir un recouvrement en arrière ou en avant dont le point cohérent se trouve dans les activités des flux parallèles ;

7. **R7** :Pour les patrons `séquence`, `diffusion parallèle`, `choix exclusif`, `choix multiples`, `jointure simple` :

   S'il y a un recouvrement en arrière les activités de ces patrons ne doivent pas être pivots.

définir des modèles transactionnels personnalisés. Malgré la pertinence de ce besoin il reste hors du centre d'intérêt de cette thèse.

4.4.4 Synthèse

Dans notre modèle transactionnel, les dépendances transactionnelles intra-activité décrivent les différentes propriétés transactionnelles des activités composantes. Les propriétés transactionnelles que nous considérons dans notre approche sont rejouable, atomique et pivot[MRKS92]. À chaque activité transactionnelle est associée un diagramme à transitions d'états qui modélise son comportement (transactionnel). Ce diagramme décrit les états possibles par lesquels une activité peut passer durant son cycle de vie et les transitions possibles entre ses états. De l'ensemble de ces états et transitions dépendent les propriétés transactionnelles de l'activité.

Les dépendances transactionnelles inter-activités spécifient les mécanismes de recouvrement. Nous en distinguons trois types : dépendance d'alternative, d'annulation et de suspension. Ces

dépendances permettent, en les combinant avec les propriétés transactionnelles, d'agir pour recouvrir un état cohérent après un échec d'exécution.

Par ailleurs, nous avons montré qu'un `flot de contrôle` fixe implicitement en partie le comportement transactionnel. À cet effet, nous avons défini un ensemble de règles définissant des relations sémantiques entre, d'une part, le `flot de contrôle` à travers les patrons de workflow, et d'autre part, le `flot transactionnel` à travers les propriétés transactionnelles et les dépendances transactionnelles des activités. Ainsi, les mécanismes de recouvrement doivent être définies conformément à ces règles.

4.5 Conclusion

Ce chapitre nous a permis de répondre à la première interrogation de notre problématique : «Comment spécifier et identifier les propriétés conceptuelles du modèle de procédés qui sont corrélées aux problèmes d'évolution et de fiabilité ?». En effet, nous avons présenté, dans ce chapitre, notre modèle de workflow transactionnel dont le but est :

- d'enrichir la description des activités des workflows et leurs dépendances pour mieux exprimer leurs propriétés transactionnelles. Ce qui permet aux concepteurs de workflows d'exploiter ces propriétés pour assurer des exécutions plus correctes et fiables.
- de combiner l'aspect coordination et l'aspect transactionnel dans un seul modèle de workflow transactionnel. Ce qui permet d'assurer un niveau de **flexibilité** élevé et de développer un ensemble de techniques pour garantir un certain degré de **correction** tout en préservant la sémantique riche appropriée aux caractéristiques des **procédés métiers** du modèle de workflow classique (c.à.d. notre modèle de workflow transactionnel préserve la capacité de modélisation des WFMS actuelles).

Pour ce faire, nous avons décrit un modèle de workflow transactionnel WfT qui permet de décrire les mécanismes de recouvrement en plus des dépendances d'exécutions <normales». Dans ce modèle de workflow transactionnel WfT, nous avons distingué en particulier entre le `flot de contrôle` et le `flot transactionnel`. Le `flot de contrôle` définit l'ordre d'exécution des activités composantes. Le `flot transactionnel` définit, à travers un modèle transactionnel, les propriétés et les dépendances transactionnelles des activités composantes. Ce flux transactionnel est sémantiquement lié au flux de contrôle.

L'originalité de notre modèle est qu'il a pu réconcilier les systèmes de workflows et les MTA, deux technologies fortes, souvent considérées concurrentes. En effet, d'une part le WfT peut être considéré comme un flot d'activités autonomes et hétérogènes. D'autre part, il peut être considéré aussi comme une transaction structurée où les activités composantes sont les sous-transactions et les interactions sont les dépendances transactionnelles. Par rapport au système de workflows, notre modèle permet d'intégrer le `flot transactionnel`, un concept qui a été (relativement) ignoré. Par comparaison au MTA, notre modèle permet des structures de contrôle plus complexes. En effet, ce modèle sert de base pour l'ensemble des techniques de découverte développées (que nous présentons dans le chapitre suivant) pour assurer une conception continue.

Par ailleurs, notons que, d'un point de vue conceptuel, nous n'avons pas introduit un nouveau langage dans notre modèle de workflow transactionnel pour décrire le comportement transactionnel. Nous avons procédé par le raffinement de notre modèle de workflow décrit par le flux de contrôle pour décrire les dépendances transactionnelles intra et les inter-activités. Ainsi le langage transactionnel que nous avons utilisé n'est qu'un raffinement du langage de workflow (c.à.d. le langage du `flot de contrôle`). En effet, les dépendances transactionnelles décrivent des états intermédiaires liés au modèle transactionnel, donnant lieu à un langage de conception de type

WF/TR. D'autre part, notons que d'un point architectural, la gestion d'échec est gérée exclusivement par le modèle transactionnel à travers la spécification des mécanismes de recouvrement, donnant lieu à une architecture de type **MTR+MWF**. Cette démarche nous permet de respecter l'objectif fixé dans notre état de l'art en combinant à la fois la classe conceptuelle **WF/TR** et la classe architecturale **MTR+MWF**.

Finalement, nous avons utilisé tout au long de ce chapitre le calcul événementiel comme outil de formalisation. Le calcul événementiel nous permet de se doter d'un outil avec un support de raisonnement et d'une base formelle «solide» permettant de construire des abstractions temporelles à différents niveaux qui sont à la fois nécessaires et adaptées aux besoins de spécifications et de preuves des techniques de découverte du chapitre suivant.

Chapitre 5

Découverte et re-ingénierie des workflows transactionnels

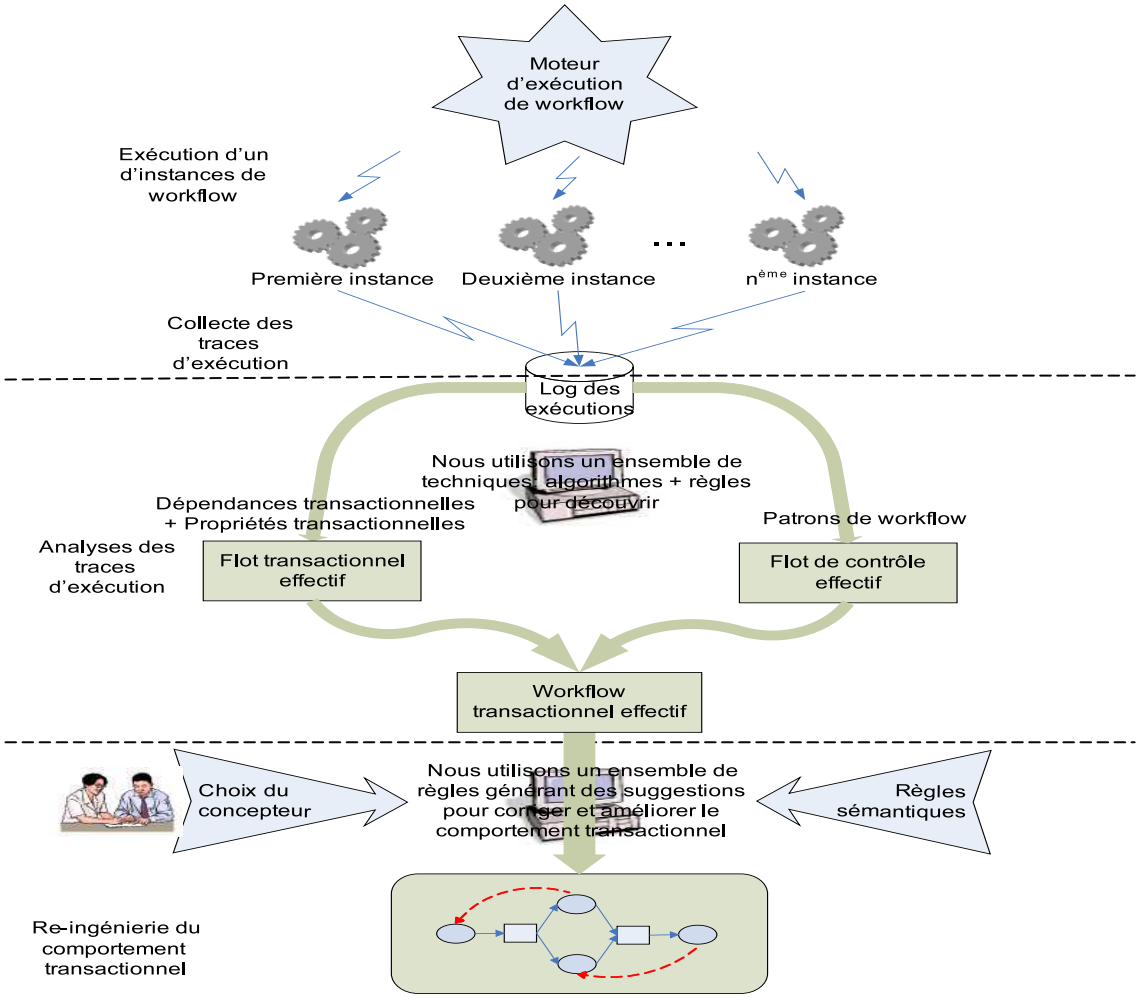
Table des matières

5.1 Introduction

Dans ce chapitre, nous décrivons un ensemble de techniques et d'algorithmes de découverte que nous avons spécifiés, démontrés et implantés pour la découverte de workflow et l'amélioration de leurs comportements transactionnels. Notre approche commence par la collecte des événements

apparus lors de l'exécution des instances de workflows. Puis, nous construisons, par des techniques statistiques, une représentation intermédiaire spécifiant des dépendances élémentaires entre les activités. Ces dépendances sont, par la suite, raffinées pour découvrir les patrons de workflow. Ensuite, nous présentons de nouvelles techniques de découverte pour extraire le comportement transactionnel de workflows et être alerté des lacunes (anomalies) de conception concernant particulièrement les mécanismes de recouvrement. Finalement, en utilisant ces résultats, nous présentons des techniques d'amélioration et/ou de correction du comportement transactionnel.

Figure 5.1 Une vue globale de notre approche de découverte et de re-ingénierie de workflow transactionnel.



Notre approche permet de détecter et de corriger des erreurs de conception dues à un omis lors de la phase de conception initiale ou une évolution du schéma de workflow observée dans la phase d'exécution. Pour ce faire, nous découvrons d'abord le workflow transactionnel effectif à partir de ses traces d'exécutions. Ensuite, nous utilisons un ensemble de règles pour améliorer et corriger son comportement transactionnel. La figure 5.1 présente une vue globale de notre approche :

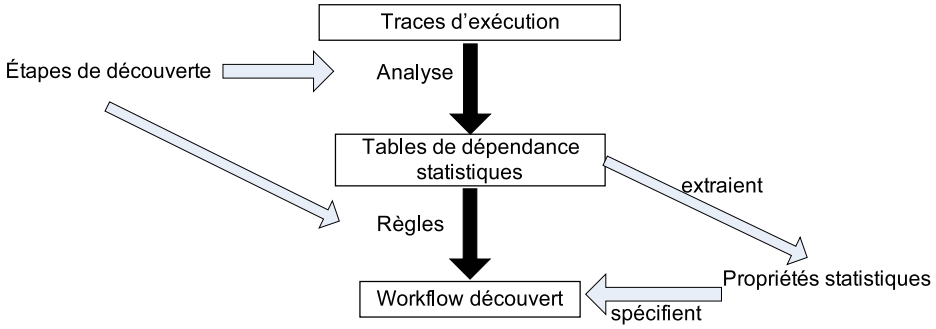
- **Collecte des traces d'exécutions du workflow :** Cette phase consiste à collecter les traces d'exécutions du workflow transactionnel à partir du moteur d'exécution du workflow.

Le nombre et la structure des traces d'exécutions doivent être suffisamment riches pour procéder à la découverte.

- **Découverte du workflow transactionnel effectif :** Le but de cette phase est de découvrir le modèle de workflow transactionnel. Il est possible, à partir des traces d'exécutions, d'extraire le `flot de contrôle`, à travers la découverte des patrons de workflow [GBG04a; GBG05; GBG06], et le `flot transactionnel` du workflow, à travers la découverte des propriétés transactionnelles des activités et du dépendances transactionnelles entre les activités [GBG04b; GG05], ce qui correspond au modèle de workflow transactionnel effectif.
- **Correction et amélioration du comportement transactionnel :** A partir des informations découvertes lors de l'étape précédente, nous utilisons un ensemble de règles générant des suggestions pour corriger et améliorer le comportement transactionnel du workflow. Ces règles dépendent des règles sémantiques définies dans le chapitre précédent et des choix métiers des concepteurs.

Pour découvrir le modèle du workflow transactionnel, que se soit le `flot de contrôle` (patrons de workflow) ou le `flot transactionnel` (dépendances transactionnelles et propriétés transactionnelles), à partir des traces d'exécutions, nous procédons en deux étapes (*c.f.* figure 5.2). La première consiste à analyser statistiquement les traces d'exécutions pour en extraire les tables de dépendances statistiques. La deuxième étape consiste à spécifier statistiquement les propriétés du `flot de contrôle` et du `flot transactionnel` sous forme de règles de découverte à appliquer au dessus des tables de dépendances statistiques.

Figure 5.2 Étapes de découverte de workflow transactionnel.



Ce chapitre est structuré comme suit. D'abord, la section 5.2 présente le modèle adopté pour les traces d'exécutions de workflows qui sera l'entrée unique pour nos algorithmes de découverte de workflow. Ensuite, la section 5.3 spécifie les analyses statistiques utilisées et détaille notre algorithme de découverte de patrons de workflows. Par la suite, la section 5.4 discute des techniques de découverte du comportement transactionnel. La section 5.4, en se basant sur les résultats découverts spécifiant le workflow effectif propose des règles de correction et d'amélioration du comportement transactionnel du workflow. Finalement, la section 5.6 conclut ce chapitre par une discussion critique de nos travaux de découverte.

5.2 Traces d'exécutions pour les workflows

Évidemment, les résultats obtenus du processus de découverte de workflow dépendent de la teneur et de la qualité des traces d'exécutions disponibles. Cette section s'intéresse aux issues relatives aux traces d'exécutions de workflows. La section 5.2.1 présente les différents travaux

entrepris pour la collecte de traces d'exécutions et les contraintes qui en découlent. Par la suite, tirant conséquences des cette analyse nous présentons, dans la section 5.2.2, notre modèle de traces d'exécutions. Nous avons essayé à travers ce modèle de donner un outil commun dans lequel nous pouvons présenter et encapsuler la plupart des formats présents dans les systèmes de gestion de workflows. Finalement, nous proposons, dans la section 5.2.3, une structure «allégée» de traces d'exécutions et les conditions minimales pour appliquer les algorithmes de découverte du `flot de contrôle`. Cette structure nous permet de couvrir la totalité des mécanismes de collecte de traces d'exécutions proposés par les systèmes de gestion de workflows indépendamment de leur hétérogénéité.

5.2.1 Contraintes et caractéristiques

Généralement, la spécification d'un workflow ne concerne pas les détails d'exécution interne des activités. Cependant elle doit traiter, au moins, des événements de changement d'états visibles des activités (tels que `initial, suspendu, activé, annulé, échoué, terminé`). Actuellement, la grande majorité des systèmes de gestion de workflows collectent divers événements se produisant pendant l'exécution des instances des workflows. N'importe quel système d'information utilisant les systèmes transactionnels tels qu'ERP, CRM, ou les systèmes de gestion de workflows offre cette information sous une certaine forme [vdAvDH+03]. Des entrepôts de données pour stocker ces traces d'exécutions ont été proposés dans la littérature scientifique [EOG02; zMM01]. Ces entrepôts de données simplifient et accélèrent les requêtes nécessaires aux techniques de découverte de workflow.

Nous supposons qu'il est possible d'enregistrer des événements tels que (i) chaque événement se rapporte à une activité (c.à.d, une étape bien définie dans le workflow), (ii) chaque événement se rapporte à un cas (c.à.d, une instance de workflows), et (iii) les événements sont totalement ordonnés. Ainsi, nous nous attendons à ce que les activités soient décelables et traçables, ce qui signifie que le workflow, doit d'une façon ou d'une autre, garder et capturer les traces d'exécutions. Pour se faire, nous avons implanté au sein de notre système de gestion de workflows Bonita un module qui nous permet de collecter les traces d'exécutions (*c.f.* le chapitre de mise en œuvre pour plus de détails).

A partir de ces traces d'exécutions, notre but est de (re)construire le modèle de workflow qui peut régénérer les mêmes traces d'exécutions. Mais, il y a cependant un certain nombre de facteurs compliquants et de contraintes auxquels nous devons faire face avant de procéder à la découverte :

- Pour de grands modèles de workflows, le processus de découverte nécessite des traces d'exécutions volumineuses. Par exemple, si le modèle contient des cheminements alternatifs et parallèles, alors des traces d'exécutions en nombre réduit ne peuvent pas contenir toutes les combinaisons possibles.
- Les traces d'exécutions de workflows peuvent contenir du bruit (c.à.d, des parties de traces d'exécutions peuvent être incorrectes ou inachevées). Des événements peuvent être collectés inexactement en raison d'erreurs humaines ou techniques. Par exemple, les événements peuvent être absents dans les traces d'exécutions si une partie de la tâche est manuelle ou manipulée par une autre unité d'organisation.
- La collecte des traces d'exécutions peut causer des problèmes légaux. Clairement, les traces d'exécutions de workflows peuvent être utilisées pour mesurer systématiquement les performances des utilisateurs. La législation en ce qui concerne des problèmes tels que la confidentialité et la protection des données personnelles diffère d'un pays à un autre. Par exemple, des compagnies françaises sont liées par «la loi personnelle de protection de don-

nées» qui est basée sur une directive de l'union européenne. Les traces d'exécutions de la plupart des systèmes de workflows contiennent des informations sur les utilisateurs, et donc, ce problème devrait être considéré minutieusement. Néanmoins, nous pouvons enlever ce type d'information sans gêner notre processus de découverte.

5.2.2 Structure des traces d'exécutions adoptée

Dans cette section, nous nous concentrons sur la syntaxe et la sémantique de l'information stockée dans les traces d'exécutions de workflows. Nous ferons ceci en présentant un format de traces indépendant de tout système. Ce format est compatible avec la plupart des systèmes de gestion de workflows. En principe, n'importe quel système qui collecte des événements liés à l'exécution de ses activités peut utiliser ce format indépendant pour stocker et échanger ses traces d'exécutions. Ce format est utilisé comme entrée unique pour les outils d'analyse et de découverte que nous allons présenter dans les sections suivantes. L'intérêt d'utiliser un format simple est de réduire l'effort de traitement et de favoriser l'utilisation des techniques de découverte dans des contextes multiples.

La trace d'exécution d'un procédé de workflow est l'ensemble des traces d'exécutions de ses instances d'exécution. Chaque instance d'exécution capture des événements atomiques (c.à.d, les événements supposés instantanés) représentant le changement d'états de ses activités suite à l'exécution d'une action. Chaque événement (*c.f.* définition 5.1) est décrit par l'identifiant unique de l'activité, le nouvel état de l'activité et le temps d'occurrence. L'ordre des événements dans une même instance d'exécution est important tandis que l'ordre différentes instances d'exécution est sans importance.

DÉFINITION 5.1 (Event)
Soit T l'ensemble d'activités appartenant à un workflow. Un événement est défini comme un tuple Event= (activityId, occurTime, activityState) où :

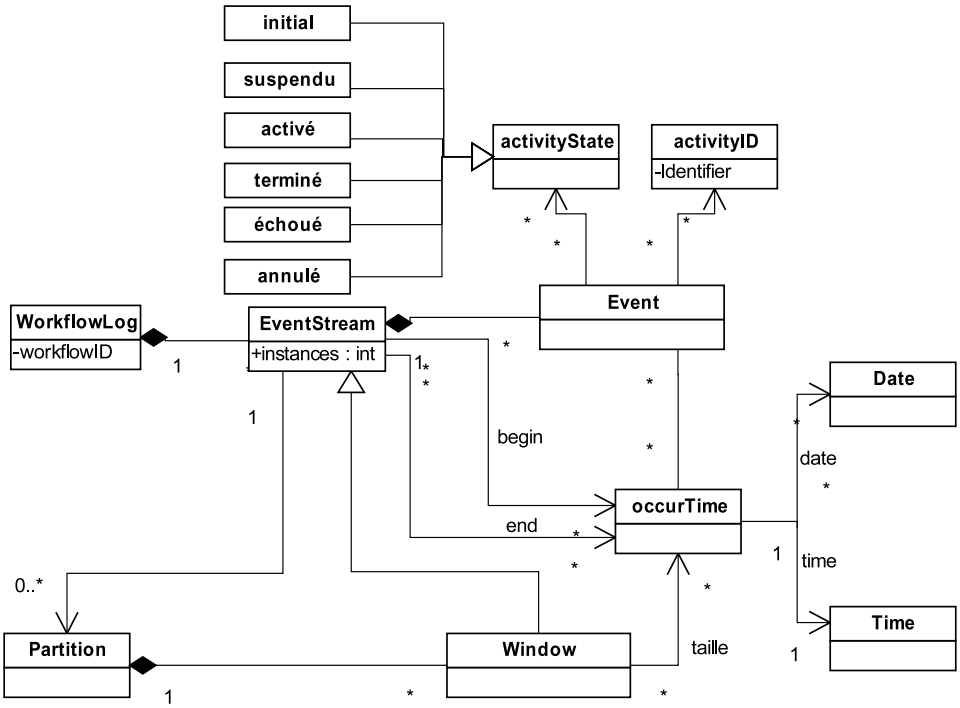
- ✓activityId : T est l'identifiant de l'activité concernée par l'événement ;
- ✓occurTime : int est l'instant d'occurrence de l'événement ;
- ✓activityState : symbol est le nouvel état de l'activité

Les événements décrivent tous les états possibles d'une activité au sein d'une instance de workflow. Quelques systèmes de collecte d'événements proposent un niveau de granulité plus fin. D'autres ne collectent qu'une partie des événements. En plus, la nomenclature est généralement différente d'un système à un autre. Par conséquent, nous avons besoin d'une uniformisation des événements. Dans notre format, nous considérons les états : `initial`, `suspendu`, `activé`, `terminé`, `échoué` et `annulé`. Ces états couvrent l'essentiel des états présents dans les différents systèmes de collecte de traces d'exécutions.

Les traces d'exécutions d'un procédé de workflow peuvent contenir des centaines voir des milliers de traces d'exécutions de ses instances. On suppose qu'il n'y a pas de lien de causalité entre ces différentes instances d'exécution. Ainsi, nous pouvons représenter les traces d'exécutions d'un workflow comme un ensemble de séquences distinctes (*c.f.* définition 5.2), tel que chaque séquence représente l'exécution d'une instance de workflow. Le diagramme de classe UML du schéma 5.3, montre qu'un WorkflowLog se compose d'un ensemble d'EventStreams. Chaque EventStream trace l'exécution d'un cas (instance) et se compose d'un ensemble d'événements (Event) qui capturent le cycle de vie des activités exécutées dans une instance.

Il est conseillé de s'assurer que les WorkflowLogs et les EventStreams sont décrits par un identifiant unique. Le nom d'activité devrait être aussi unique dans le workflow. S'il y a deux activités

Figure 5.3 Un modèle d'une trace d'exécution d'un Workflow



ou plus dans un workflow ayant le même nom, on assume qu'elles se rapportent à la même activité. Bien qu'on suppose qu'une activité ait un nom unique, une activité peut être présente plusieurs fois dans la même instance. Ce qu'on peut observer par exemple dans une boucle.

Ci-dessous nous avons un EventStream relatif à une instance d'exécution du workflow exemple de la figure 2.5.

$\mathbf{L}$ = EventStream((13/5/2005,5 :42 :12), (14/5/2005, 14 :01 :54), [Event("STP", `initial`, (13/5/2005, 5 :42 :12)), Event("VSC", `initial`,(13/5/2005, 5 :42 :12)), Event("VD", `initial`, (13/5/2005, 5 :42 :12)), Event("ER", `initial`, (13/5/2005, 5 :42 :12)), Event("MJR", `initial`, (13/5/2005, 5 :42 :12)),Event("AE", `initial`, (13/5/2005, 5 :42 :12)),Event("RP", `initial`, (13/5/2005, 5 :42 :12)),Event("RD", `initial`, (13/5/2005, 5 :42 :12)), Event("LD", `initial`, (13/5/2005, 5 :42 :12)), Event("STP", `activé`, (13/5/2005, 5 :42 :12)), Event("STP", `terminé`, (13/5/2005, 5 :43 :12)), Event("VSC", `activé`, (13/5/2005,7 :11 :12)), Event("VD", `activé`, (13/5/2005,7 :11 :12)), Event("VSC", `terminé`, (13/5/2005,11 :11 :12)), Event("ER", `activé`, (13/5/2005,11 :11 :12)), Event("ER", `échoué`, (13/5/2005,14 :01 :54)), Event("AE", `activé`, (13/5/2005, 17 :20 :01)), Event("AE", `suspendu`, (13/5/2005, 18 :00 :01)), Event("AE", `activé`, (14/5/2005, 00 :00 :01)), Event("AE", `terminé`, (14/5/2005, 00 :01 :54)), Event("VD", `terminé`, (14/5/2005,5 :45 :54)), Event("RD", `activé`, (14/5/2005,8 :00 :24)), Event("RD", `terminé`, (14/5/2005,10 :32 :55)), Event("LD", `activé`, (14/5/2005,12 :00 :24)), Event("LD", `terminé`, (14/5/2005,14 :01 :54))])

L'expérience montre qu'il est également assez simple d'extraire l'information à partir des systèmes d'information d'entreprise spécifiques et de le traduire au format XML. Le format que nous avons présenté peut être facilement transcrit au format XML proposé et utilisé par plusieurs outils de découverte de workflow [vDdMV+05]. La figure 5.4 montre la structure XML

DÉFINITION 5.2 (WorkflowLog)
Un EventStream représente l'historique d'exécution, en termes d'événements, d'une instance de workflow comme un tuple EventStream= (begin, end, sequenceLog, instances) où :

✓begin : TimeStamp est l'instant du début de la trace d'exécution de cette instance ;

✓end : TimeStamp est l'instant de la fin de la trace d'exécution de cette instance ;

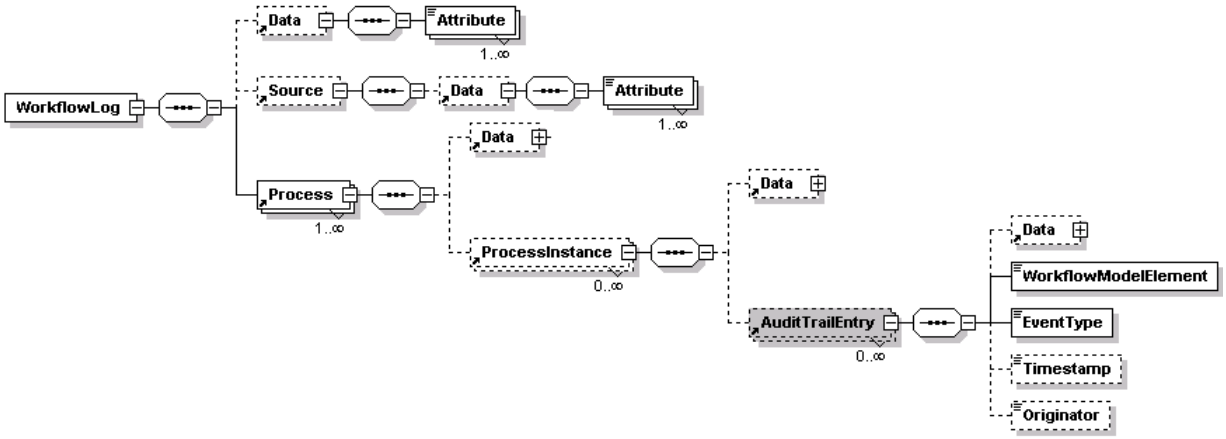
✓sequenceLog : Event* est un ensemble ordonné d'événements collectés à l'exécution de cette instance de workflow ;

✓instances : int est le numéro de la trace d'exécution de cette instance.

Un WorkflowLog est un ensemble de EventStreams. WorkflowLog=(workflowID, {EventStream$_i$, $0 \leq$ i $<$ nombre d'instances de workflow}) où EventStream$_i$ est la trace d'exécution de la ième instance du workflow.

de ce format.

Figure 5.4 Structure XML des traces d'exécutions



5.2.3 Conditions minimales sur les traces d'exécutions

Un grand nombre d'applications de collecte de traces d'exécutions de workflows contient une estampille de temps pour chaque événement et cette information peut être très utile pour extraire une information additionnelle en utilisant les états `activé` et `terminé` pour détecter explicitement le parallélisme. Les états `activé` et `terminé` peuvent également être utilisés pour extraire le temps d'exécution d'une activité. En plus, un certain nombre de traces d'exécutions contiennent également des informations sur d'autres états «transactionnels» de l'activité offrant la possibilité d'analyser le comportement transactionnel du workflow.

Cependant, certains systèmes de collecte d'événements proposent un niveau de granularité moins fin et ne collecte qu'une partie des états. En plus, les nomenclatures des états sont généralement différentes d'un système à un autre. Par exemple, le système de gestion de workflows Staffware ne collecte que les états `suspendu`, `abandonné` et `terminé`. Ces états sont nommés respectivement «Processed To», «Withdrawn», and «Released By». Staffware n'enregistre pas les autres états tels que : `initial`, `activé` et `échoué`. Mais il enregistre des événements additionnels

qui ne se trouvent pas dans notre format, (comme «Expired», par exemple). Comme solution, nous pouvons simplement choisir de les filtrer sinon d'affecter un état par défaut, lors d'un mappage des traces d'exécutions de Staffware au format que nous avons proposé dans la section précédente. Cependant dans le cas du processus de découverte de `flot de contrôle`, parfois il est plus pratique de considérer simplement les activités en tant qu'événements atomiques, qui ne prennent pas de temps d'exécution et qui présentent seulement l'événement de terminaison avec succès, pour simplifier la présentation et minimaliser les contraintes de collecte de traces d'exécutions.

En effet, nous nous s'intéressons, dans le `flot de contrôle` d'un workflow (*c.f.* chapitre précédent), uniquement aux instances exécutées avec succès sans échec ou exception. Ainsi, les traces d'exécutions des instances utilisées pour la découverte de flot de contrôle présentent toutes des activités qui ont atteint l'état `terminé` sans exception ou échec. En conséquence, nous pouvons présenter, pour la découverte du `flot de contrôle`, un format de traces d'exécutions d'une instance de workflow simplifié (*c.f.* définition 5.3), sans vraiment perdre de généralité, où les événements sont atomiques et totalement ordonnés ne contenant que le nom de l'activité. Il n'y a aucun besoin d'utiliser d'autres EventStreams relatives aux échecs d'exécutions. En fait, ces cas concernent seulement le comportement et les dépendances transactionnelles et seront utilisés par la suite pour la découverte du comportement et des dépendances transactionnelles qui désignent les mécanismes pour le traitement des échecs.

DÉFINITION 5.3 (WorkflowLog SIMPLIFIÉ)
Soit $\mathcal{AT}$ l'ensemble des activités transactionnelles. $\sigma \in \mathcal{AT}^*$ est un EventStream simplifié représentant un ensemble d'activités exécutées avec succès et ordonnées selon leur ordre d'exécution. Nous notons par $\text{WorkflowLog}_{termine}$ les traces d'exécutions simplifiées d'un workflow regroupant l'ensemble des EventStreams simplifiés.
Nous notons par $WL \in \mathcal{P}(\mathcal{AT}^*)$ l'ensemble des traces d'exécutions simplifiées.

Comme les traces d'exécutions initialement collectées contiennent d'autres états, nous allons procéder à un filtrage du WorkflowLog pour ne conserver que les traces d'exécutions d'instances de workflow exécutées avec succès. Ainsi, l'unique condition à la découverte du flot contrôle est d'avoir des traces d'exécutions qui rapportent l'état `terminé` de leurs activités. Cette caractéristique «minimaliste» nous permet d'exploiter des traces d'exécutions «pauvres» qui ne contiennent que l'information concernant la succession des activités exécutées avec succès sans rapporter par exemple les états d'exécution intermédiaires des activités ou les temps d'exécution.

Cependant pour découvrir le comportement parallèle en l'absence d'indication sur le temps d'exécution des activités, nous supposons aussi que deux activités sont en parallèle si elles apparaissent sans ordre de précédence dans les traces d'exécutions. Ceci implique que notre approche ne fonctionnera qu'à la condition que les traces d'exécutions respectent cette contrainte en cas de comportement parallèle. Cette exigence s'inscrit dans la première condition de l'axiome 5.1 spécifiant les caractéristiques décrivant les propriétés des traces d'exécutions nécessaires à notre approche de découverte du `flot de contrôle`. En supposant que les événements sont enregistrés à l'heure de leur occurrence, cet axiome regroupe l'ensemble des conditions pour qu'une structure de traces d'exécutions «minimaliste» définisse des traces d'exécutions complètes d'un workflow.

La deuxième condition de l'axiome 5.1 indique que des traces d'exécutions de workflow complètes doivent remplir la condition : si une activité précède une autre dans le `flot de contrôle` alors il doit y avoir une trace d'exécution d'une instance de workflow rapportant deux événements relatifs à ces deux activités qui se suivent. En particulier, la troisième condition spécifie

AXIOME 5.1 (AXIOME DE TRACES D'EXÉCUTIONS COMPLÈTES)
$wft = (WA, Ax)$ un workflow transactionnel et $L \in WL$ la trace d'exécution de wft : L est complète si et seulement si :

1. $\forall a, b \in WA$ tel que a et b sont en concurrence, alors $\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1}, \sigma_1 = q_1 q_2 q_3 ....q_{m-1} \in L \wedge \exists 0 < i < n, 0 < j < m | t_i = a, t_{i+1} = b \wedge q_j = b, q_{j+1} = a$
2. $\forall a, b \in WA$ tel que $a \prec b$, alors $\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < j < n, | t_i = a, t_j = b$
3. $\forall a, b \in WA$ tel que $a \prec b \wedge (\texttt{OUT}_{max}(a) = \texttt{OUT}_{min}(a) = 1 \vee \texttt{IN}_{max}(b) = \texttt{IN}_{min}(b) = 1)$, alors $\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < n, | t_i = a, t_{i+1} = b$

que si l'exécution d'une activité dépend directement de la terminaison d'une autre activité, alors deux événements relatifs à ces deux activités doivent se suivre directement (immédiatement, sans événement intercalé entre elles) au moins une fois dans une trace d'exécution d'une instance. En d'autre termes, pour être complètes, les traces d'exécutions des workflows doivent couvrir tous les cas possibles (c.à.d si un élément spécifique décrivant un élément de routage au niveau du `flot de contrôle`, les traces d'exécutions doivent contenir ce comportement au moins une fois dans la trace d'exécution d'une instance).

Nous pouvons en déduire facilement à partir de l'axiome de traces d'exécutions complètes le nombre d'instances suffisant pour des traces d'exécutions complètes pour un workflow donné (*c.f.* lemme 5.1). Ce lemme indique pour chaque comportement, le nombre d'instances nécessaires à notre technique de découverte du `flot de contrôle`. Cette faculté définit une «spécification locale» sur le nombre d'instances pour des traces d'exécutions complètes. Grâce à cette «spécification locale», nous n'exigeons pas la présence de toutes les instances d'exécution possibles pour un workflow pour satisfaire la contrainte de traces d'exécutions complètes. Par exemple, dans un workflow qui contient n activités concurrentes qui se suivent par m activités concurrentes le nombre de scénarios possibles (nombre de traces d'exécutions d'instances différentes) est égal a $n! * m!$. Mais, en appliquant notre lemme de nombre d'instances suffisantes pour des traces d'exécutions complètes nous n'avons besoin que de $n! + m!$. Notons q'une preuve détaillée du lemme 5.1 se trouve dans l'annexe B.

LEMME 5.1 (NOMBRE D'INSTANCES POUR DES TRACES D'EXÉCUTIONS COMPLÈTES)
Soit wft un workflow transactionnel le nombre minimal et suffisant de traces d'exécutions d'instances différentes pour la découverte du `flot de contrôle` de wft est défini comme suit :

1. Le nombre est égal à 1 pour les workflows ne contenant qu'une suite séquentielle d'activités sans comportement concurrentiel ni conditionnel ;
2. Un comportement conditionnel entre n activités avant un point de «jointure» ou après un point de «diffusion» nécessite n traces d'exécutions d'instances différentes reprenant à chaque fois un choix.
3. Un comportement concurrentiel entre n flux de contrôle, chaque flux i ; $0 < i < n + 1$ contient j_i activités nécessite $\Pi_{i=1..n}(j_i + i - 1) = j_1 * (j_2 + 1) * (j_3 + 2) * (j_4 + 3) * .....* (j_n + n - 1)$ traces d'exécutions d'instances différentes reprenant toutes les combinaisons possibles.

Le tableau 5.2 représente l'exécution sans échec de sept instances de notre exemple de workflow de prêt bancaire. Ce tableau contient l'information suffisante que nous supposons être présente pour la découverte du `flot de contrôle`. En effet, notre workflow exemple contient d'une part

Identifiant de l'instance	EventStream
Instance1	STP VSC ER VD MJR RD LD
Instance2	STP VSC AE VD RD LD
Instance3	STP VSC VD RP RD LD
Instance4	STP VSC VD ER MJR RD LD
Instance5	STP VSC ER MJR VD RD LD
Instance6	STP VSC VD AE RD LD
Instance7	STP VSC RP VD RD LD
Instance8	STP VD VSC RP RD LD
Instance9	STP VSC RP RD VD LD

TAB. 5.1 – Traces d'exécutions de 7 instances du workflow du prêt bancaire

un comportement conditionnel entre trois activités (ER, AE, et RP), ce qui nous fait 4 instances (1+3), si on ajoute l'instance nécessaire pour satisfaire le premier point de l'axiome. Et d'autre part, 2 flux de contrôles concurrents contenant respectivement une activité (VD), et 4 activités au maximum (VSC, (ER ou AE ou (RP et MJR)) et RD) concurrentes ce que nous fait 1*(4+1)= 5 instances. Le total d'instances suffisantes sera ainsi égale à 5+4 = 9, ce que le tableau 5.1 rapporte.

Dans ce tableau, les instances 1, 4 et 5 décrivent le cas ou le prêt bancaire a été traité selon la procédure normale. Bien que ces différentes instances (1, 4 et 5) décrivent le même traitement, le scénario d'exécution des activités concurrentes n'est pas le même (c.à.d les activités ER, VD et MJR n'ont pas le même ordre de terminaison à chaque fois). Ces différentes instances, (de même pour les instances 2, 6, d'une part, et 3, 7, 8 et 9, d'autre part) nous permettent à la fois de décrire les différents choix possibles du traitement du prêt ainsi que les différents scénarios possibles d'exécution des activités concurrentes dans ces choix.

5.2.4 Synthèse

Dans cette section, nous avons discuté des issues et des contraintes relatives aux mécanismes et à la structure des traces d'exécutions de workflow. Nous avons adopté une structure unique qui est partagée et conforme à la plupart des systèmes ou mécanismes de collecte de traces d'exécutions de workflows actuels. Ces traces d'exécutions seront l'unique entrée pour nos techniques de découverte de workflow. Ainsi, nous avons discuté des conditions minimales et suffisantes que ces traces d'exécutions doivent assurer pour nos techniques de découverte.

Nous nous sommes intéressés, en particulier aux contraintes du processus de découverte de `flot de contrôle`. Nous avons défini, d'une part, une structure minimale décrivant la nature ou la «qualité» d'information, et d'autre part, le nombre ou la «quantité» relative d'instances de workflow nécessaire et suffisante au processus de découverte. En conclusion, cette section nous a permis de traiter de la deuxième interrogation de notre problématique : «Quelles contraintes sur la structure et la richesse des données de traces d'exécutions sont nécessaires au processus de découverte de procédés ?».

Dans la section suivante, nous allons détailler les techniques d'analyse statistique appliquées au dessus de ces traces d'exécutions simplifiées et complètes pour découvrir les patrons de workflow, c.à.d le `flot de contrôle`.

5.3 Découverte du flot de contrôle

5.3.1 Vue générale

Nous commençons par construire par des techniques statistiques, une représentation intermédiaire modelant des **dépendances élémentaires** en analysant les traces d'exécutions (*c.f.* section 5.3.2). Ces dépendances sont raffinées par la suite (*c.f.* section 5.3.3) sous la forme de propriétés statistiques qui capturent les principaux comportements du flot de contrôle. Dans la section 5.3.4, nous exploitons ces propriétés statistiques pour découvrir des structures avancées de **patrons de workflow**. En effet, un **patron de workflow** est un ensemble de **dépendances élémentaires** qui définit une structure avancée exprimant un comportement spécifique, en termes de flot de contrôle, caractérisé par les propriétés statistiques.

Comme il est illustré par la figure 5.5, nous adaptons une approche ascendante dans notre processus de découverte de patrons de workflow.

1. **La découverte des dépendances entre activités :** D'abord, nous spécifions les dépendances liant les activités du workflow pendant l'exécution. Nous divisons ces dépendances en deux catégories : causales et non-causales.

   Une dépendance causale entre deux activités exprime le fait que l'occurrence de la première activité implique l'activation de la deuxième activité. Cette dépendance causale, dans le cadre du flot de contrôle, représente la dépendance d'exécution normale décrite dans le chapitre précédent (*c.f.* définition 4.6). Tandis qu'une dépendance non-causale spécifie tout autre type de dépendance comportementale exprimant une propriété non-causale du flot de contrôle (le comportement parallèle par exemple).

2. **Le calcul des propriétés statistiques de comportement :** Deuxièmement, nous spécifions les propriétés statistiques du flot de contrôle. Ces propriétés façonnent les comportements de base des patrons de workflow à découvrir. Nous définissons trois types de propriétés : «séquentielles», «parallèles» et «conditionnelles».

   Les propriétés «séquentielles» et «conditionnelles» héritent de la dépendance causale. La première exprime une dépendance causale exclusive entre deux activités. Tandis que la seconde définit un ensemble de dépendances causales entre, d'une part, une activité, et d'autre part, une ou plusieurs activités d'un ensemble d'activités permettant de décrire et de spécifier un choix entre ces activités. La propriété «parallèles» hérite de la dépendance non-causale et caractérise le comportement concurrent entre un ensemble d'activités.

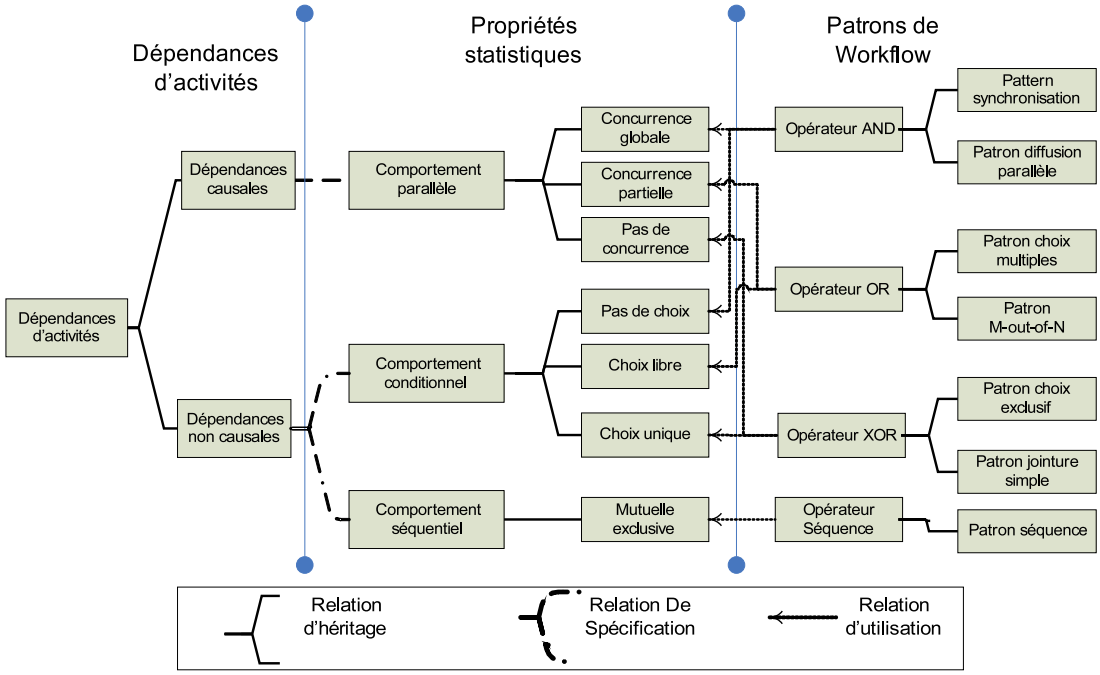
3. **La découverte des patrons de workflow :** Finalement, nous utilisons un ensemble de règles pour découvrir les patrons de workflow. Ces règles s'expriment à l'aide de propriétés statistiques qui déterminent d'une manière unique le patron découvert.

   Dans ce travail, nous avons choisi de découvrir les 7 patrons de workflow les plus communs et les plus utilisés, mais l'approche adoptée offre la possibilité d'enrichir cet ensemble de patrons en indiquant de nouvelles dépendances et/ou de nouvelles propriétés statistiques associées ou en utilisant les propriétés existantes dans des combinaisons découvrant de nouveaux patrons.

5.3.2 Découverte des dépendances élémentaires

Le but de cette section est de décrire notre algorithme pour la découverte des dépendances élémentaires à partir du WorkflowLog et d'établir un modèle intermédiaire représentant ces dépendances : la table de dépendances statistiques (ou TDS).

Figure 5.5 Une vue hiérarchique de notre approche pour la découverte de patrons de workflow



5.3.2.1 Découverte des dépendances directes

Une dépendance élémentaire est un dépendance «immédiate» ou «directe»[23] liant deux activités dans le sens où un événement de la première précède directement celui de l'autre dans un EventStream. Afin de découvrir les dépendances directes dans un WorkflowLog, nous avons besoin d'une représentation intermédiaire statistique. Cette représentation intermédiaire se présente sous la forme d'une table de dépendances statistiques (ou TDS) qui se base sur la notion de table de fréquences comme présentée dans [CW98c]. Comme les patrons de workflow sont décrits uniquement par des dépendances spécifiant le `flot de contrôle`, cette table capture les dépendances directes du `flot de contrôle` qui sont relatives exclusivement aux événements `terminé` se trouvant dans un $\text{WorkflowLog}_{termine}$.

La TDS est construite à travers un calcul statistique qui extrait les dépendances élémentaires entre les activités d'un $\text{WorkflowLog}_{termine}$ comme il est défini dans la définition 5.3. Pour chaque activité[24] A, nous extrayons à partir de $\text{WorkflowLog}_{termine}$ l'information (les statistiques) suivante : (i) le nombre d'occurrences global de cette activité (dénotée $\#A$) et (ii) les dépendances directes aux activités précédentes B_i (dénoté $P(A/B_i)$). La taille de TDS est $n * n$, où n est le nombre d'activités dans le workflow. L'entrée (i,j) de la table (notation $P(A_{0\leq i<n}/A_{0\leq j<n})$) est la fréquence de la j^{ieme} activité qui précède immédiatement la i^{ieme} activité. La table illustrée dans le tableau 5.2 représente la TDS de notre exemple de workflow de prêt bancaire. Par exemple, dans cette table P(MJR/ER)=0.69 exprime que l'événement correspondant à l'exécution de ER à 69% de chance de se produire immédiatement avant l'événement correspondant à l'exécution de MJR dans les traces d'exécutions du workflow.

Le théorème 5.1 démontre une corrélation entre les dépendances normales inter-activités et

[23] Les termes *immédiate* et *directe* seront interchangeables dans la suite

[24] Nous faisons ici une confusion entre activité et événement correspondants dans les traces d'exécutions car on a un seul type d'événement par activité correspondant à l'état `terminé`

P(x,y)	STP	VSC	ER	MJR	AE	RP	RD	VD	LD
STP	0	0	0	0	0	0	0	0	0
VSC	**0.94**	0	0	0	0	0	0	0.06	0
ER	0	**0.71**	0	0	0	0	0	0.29	0
MJR	0	0	**0.67**	0	0	0	0	0.33	0
AE	0	**0.87**	0	0	0	0	0	0.13	0
RP	0	**0.92**	0	0	0	0	0	0.08	0
RD	0	0	0	0.48	0.22	0.18	0	0.12	0
VD	**0.06**	0.1	0.14	0.2	0.04	0.06	0.4	0	0
LD	0	0	0	0	0	0	**0.21**	**0.79**	0

$$\#STP = \#VSC = \#VD = \#RD = \#LD = 100$$
$$\#ER = \#MJR = 69\,;\, \#AE = 21\,;\, \#RP = 10$$

TAB. 5.2 – Table Statistique de Dépendances initiale ($P(x/y)$) et les fréquences des activités (#)

les statistiques des traces d'exécutions exprimées dans TDS. Ainsi chaque dépendance normale entre deux activités du workflow s'exprime par une valeur positive dans l'entrée correspondante dans TDS. Nous notons respectivement «$\prec_{TDS}$» et «$\lhd_{TDS}$» les relations de dépendances directes et indirecte induites par TDS [25], par symétrie aux relations de dépendances de la définition 4.8.

THÉORÈME 5.1 (CORRÉLATION ENTRE TDS ET LES DÉPENDANCES D'ACTIVITÉS)
Soit $wft = (WA \subset \mathcal{AT}, Ax \subset \mathcal{A}xiomes)$ un workflow transactionnel où les axiomes décrivent des patrons de workflow inclus dans les 7 patrons énoncés. $\forall a, b \in WA$ tel que $a \prec b \Rightarrow P(b/a) > 0$.

Dans notre approche, nous traitons des modèles de workflow ne contenant pas de boucle de longueur égale à deux. Ainsi il est facile de déduire le lemme 5.2. Ce lemme exprime une relation d'équivalence entre les entrées positives dans TDS et les dépendances entre les activités concernées. Notons q'une preuve détaillée de ce lemme et du théorème 5.1 se trouve dans l'annexe B.

Mais cette TDS, comme elle est calculée, présente quelques problèmes pour exprimer «correctement» et «complètement» les dépendances d'activités concernant le comportement parallèle et conditionnel des activités. En effet, ces entrées ne permettent pas de pouvoir identifier pertinemment ces comportements. Dans la suite, nous détaillons ces problèmes et nous proposons des solutions pour corriger et compléter ces statistiques, afin de pouvoir les exploiter pour découvrir les patrons de workflow.

LEMME 5.2 (CORRÉLATION ENTRE TDS ET LES DÉPENDANCES INTER-ACTIVITÉS)
soit $wft = (WA \subset \mathcal{AT}, Ax \subset \mathcal{A}xiomes)$ un workflow transactionnel sans boucle de taille égale à 2 où les axiomes décrivent des patrons de workflow inclus dans les 7 patrons énoncés. $\forall a, b \in WA$ tel que $a \prec b \Leftrightarrow P(b/a) > 0 \wedge P(a/b) = 0$

[25] a $\prec_{TDS}$ b $\Leftrightarrow$ P(b/a)>0

5.3.2.2 Élimination des dépendances erronées

Si nous supposons que chaque EventStream contenu dans WorkflowLog vient d'un workflow séquentiel (c.à.d, ne possédant pas un comportement concurrentiel), une entrée égale à zéro dans TDS représente une indépendance causale, et symétriquement une entrée différente de zéro signifie une relation causale de dépendance (c.à.d, une relation séquentielle ou conditionnelle). Mais, en cas de comportement concurrent, comme on peut le voir dans les patrons de workflow (par exemple, *diffusion parallèle*, *synchronisation*, etc.), des EventStreams peuvent contenir des séquences d'événements entrelacées venant de flux concurrents. Par conséquence, quelques entrées dans la TDS initiale peuvent indiquer des valeurs non nulles qui ne correspondent pas à des dépendances causales. Par exemple, l'EventStream 3 donné dans le tableau 5.1 «suggère» des dépendances causales incorrectes entre VSC et VD d'une part, et entre VD et RP d'autre part. En effet, l'activité VSC est exécutée juste avant l'activité VD et VD est exécutée juste avant l'activité RP dans cet EventStream. Ces entrées sont incorrectes parce qu'il n'y a aucune dépendance causale entre ces activités comme le tableau 5.2 le suggère dans les entrées $P(VSC/VD)$ et $P(VD/VSC)$ (qui sont différents de zéro). Les valeurs soulignées dans ce tableau rapportent ce comportement pour d'autres cas similaires.

Formellement, en se basant sur le premier point du lemme 5.1 nous pouvons facilement déduire que deux activités A et B sont concurrentes si et seulement si les entrées $P(A/B)$ et $P(B/A)$ dans TDS sont non nulles. En se basant sur cette déduction, nous proposons un algorithme pour découvrir le parallélisme et marquer les entrées incorrectes dans TDS. Par ce marquage, nous pouvons éliminer la confusion provoquée par le comportement concurrentiel produisant ces entrées non nulles et erronées. L'algorithme 1 parcourt (scanne) la TDS initiale et marque les dépendances d'activités concurrentes en changeant leurs valeurs par (-1). Par exemple, nous pouvons déduire du table 5.2 que les activités VSC et VD sont en concurrence (c.à.d, $P(VSC/VD) \neq 0 \wedge P(VD/VSC) \neq 0$)), ainsi après application de notre algorithme $P(VSC/VD)$ et $P(VD/VSC)$ seront égal à (-1). Le résultat de l'application de cet algorithme sera contenu dans une table intermédiaire que nous appelons la TDS marquée.

```
   Entrées : TDS : Table de dépendances statistiques
   Sorties : TDSM : Table de dépendances statistiques marquées
   Données : TDSM_size : int ;
1  début
2  |  TDSM=TDS ;
3  |  TDSM_size = Size-tab(TDSM) ; /*retourne la taille de TDSM*/
4  |  pour int i=0 ; i<TDSM_size ; i++ ; faire
5  |  |  pour int j=0 ; j<i ; j++ ; faire
6  |  |  |  si TDS[i][j] > 0 ∧ TDS[j][i]>0 alors
7  |  |  |  |  TDSM[i][j] =-1 ;
8  |  |  |  |  TDSM[j][i] =-1 ;
9  fin
```

Algorithme 1 : Marquage des activités concurrentes dans TDS

5.3.2.3 Découverte des dépendances indirectes

À cause du comportement concurrentiel, l'exécution d'une activité pourrait ne pas dépendre de son prédécesseur immédiat dans l'EventStream, mais elle pourrait dépendre d'autres activités qui la précèdent «indirectement». Par exemple dans l'EventStream 1 donné dans le tableau 5.1, la terminaison de VD arrive entre ER et MJR. Par conséquence, ER ne se termine pas toujours juste avant MJR dans les traces d'exécutions du workflow. Ainsi, nous avons $P(MJR/ER) = 0.67$ qui présente une fréquence de dépendance sous estimée. En fait, la bonne valeur doit être égale à 1 parce que l'exécution de MJR dépend exclusivement de ER. De même, les valeurs en gras dans la TDS initiale rapportent un comportement similaire à ce que nous venons de décrire.

DÉFINITION 5.4 (FENÊTRE D'ACTIVITÉS CONCURRENTES)
Une fenêtre d'activités concurrentes définit une tranche (intervalle) d'événements dans un EventStream S : **stream** (bStream, eStream, sLog, workflowocc). Formellement, on définit la fenêtre de concurrence comme un triplet **window**(wLog, bWin, eWin) :
- (bWin : TimeStamp) et (eWin : TimeStamp) sont les moments de début ou de fin de la fenêtre (avec bStream ≤ bWin et eWin ≤ eStream)
- wLog ⊂ sLog et ∀ e : **event** ∈ S.sLog où bWin ≤ e.TimeStamp ≤ eWin ⇒ e ∈ wLog.

Nous définissons la fonction *width*(**window**) qui retourne la taille de la fenêtre d'activités concurrentes **window**.

```
   Entrées : TDSM : Table de dépendances statistiques marquées
   Sorties : TTFC : la table de tailles des FAC
   Données : TDSM_size : int ;
 1 début
 2     TDSM_size = Size-tab(TDSM) ; /*retourne la taille de TDSM*/
 3     pour int i=0 ; i<TDSM_size ; i++ ; faire
 4         ACWT[i]=1 ;
 5     pour int i=0 ; i<TDSM_size ; i++ ; faire
 6         pour int j=0 ; j<TDSM_size ; j++ ; faire
 7             si TDSM[i][j] =-1 alors
 8                 TTFC[i]++ ;
 9                 TTFC[j]++ ;
10                 pour int k=0 ; k<TDSM_size ; k++ ; faire
11                     si TDSM[k][i] >0 alors
12                         TTFC[k]++ ;
13 fin
```

Algorithme 2 : Calcul de la taille de la fenêtre de Concurrence

Pour découvrir ces dépendances indirectes, nous spécifions la notion de «fenêtre d'activités concurrentes» (*c.f.* définition 5.4). Une fenêtre d'activités concurrentes (FAC) est liée à l'activité correspondant à son dernier événement et couvre les événements des activités qui la précédent directement et indirectement. Au début, la largeur de la FAC d'une activité est égale à 1. Chaque fois que cette activité est en concurrence avec une autre activité nous ajoutons 1 à la taille de la fenêtre. Si cette activité n'est pas en concurrence avec d'autres activités et a des activités

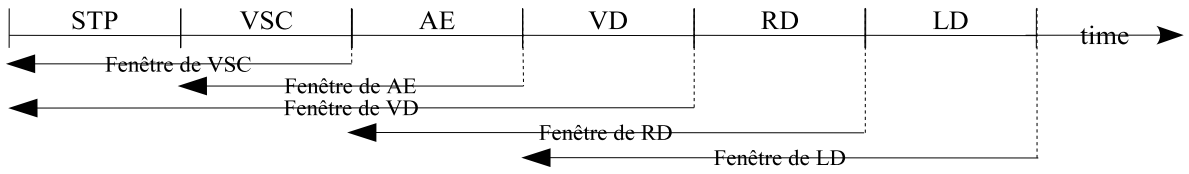
concurrentes qui la précédent, alors nous ajoutons leur nombre à la taille de sa FAC. Par exemple, l'activité RD est en concurrence avec VD alors la largeur de sa FAC est égale à 2. Se Basant sur ceci, l'algorithme 2 calcule pour chaque activité la taille de FAC qui les regroupe dans la table de tailles des FAC (TTFC) qui indique pour chaque activité la taille de sa FAC. Cet algorithme scanne la TDS marquée, calculée dans la dernière section, et met à jour TTF en fonction du comportement concurrentiel.

Ensuite, nous procédons par une partition (*c.f.* figure 5.3) de chaque EventStream qui construit un ensemble de fenêtres se chevauchant partiellement en se basant sur la TTFC. La définition 5.5 spécifie que chaque fenêtre partage l'ensemble de ses éléments avec la fenêtre qui la précède sauf le dernier événement qui contient l'activité référence de la fenêtre.

DÉFINITION 5.5 (PARTITION)
Une **partition** construit un ensemble de fenêtres qui se chevauchent partiellement au dessus d'un EventStream.
Partition : WorkflowLog → (Window)*
S : EventStream(sLog, workflowocc, bStream, eStream) → $\{w_i$:Window ; 0≤i<n$\}$:
- w_1.bWin = bStream and w_n.eWin = eStream,
- $\forall w$: *window* $\in$ *partition*, e :Event= le dernier événement dans w, *width*(w)= ACWT[e.ActivityID],
- $\forall$ 0≤i<n ; w_{i+1}.wLog - {le dernier événement e :Event dans w_{i+1}.wLog$\} \subset w_i$.wLog $\wedge$ w_{i+1}.wLog $\neq w_i$.wLog.

Dans le schéma 5.6, nous avons appliqué une partition au-dessus de l'EventStream 2 du tableau 5.1. Par exemple, la taille de la FAC de VD est égale à 4 parce que cette activité est en concurrence avec trois activités précédentes en concurrence AE, VSC et RD. Et, la taille de la FAC de RD est égale à 2 parce que cette activité est en concurrence seulement avec VD. Notons que pour chaque activité dans cet EventStream son FAC lui permet de couvrir toutes, et seulement toutes, les activités qui sont dans ses pré-conditions d'exécution.

Figure 5.6 Une partition d'EventStream



Finalement, l'algorithme 3 calcule la TDS finale. Pour chaque FAC, il calcule pour sa dernière activité les fréquences des activités qui la précèdent. La TDS finale est calculée en divisant les entrées de chaque rangée par la fréquence de l'activité qui lui est relative.

Maintenant, en appliquant ces algorithmes, nous pouvons calculer la TDS finale (*c.f.* tableau 5.3) qui sera utilisée dans la section suivante pour découvrir les patrons de workflow.

Notons que notre approche s'adapte dynamiquement au comportement concurrentiel, à travers la taille du FAC. En effet, cette taille est sensible au comportement concurrentiel : elle augmente en cas de concurrence et elle est «neutre» (égale à 1) en cas d'absence de comportement concurrentiel.

Entrées : Wlog : WorkflowLog ;
: Table d'occurrences des activités ;
$TDSM$: Table de dépendances statistiques marquées ;
Sorties : $TDSMF$:Table de dépendances statistiques finale
Données : $a_{reference}$: int ;
$a_{preceded}$: int ;
fWin : window ;
depFreq :int[][] ; /*initialisé à 0 ;*/
freq :int ;

```
1  début
2      TDSM_size = Size-tab(TDSM) ;
3      pour tout win :window dans partition(Wlog) faire
4          a_reference = last-activity(win) ; /* la fonction last-activity renvoie la dernière
           activité du dernier événement de win*/ win = preceded-events(win) ; /* la
           fonction last-activity renvoie win sans le dernier événement*/
5          pour int i=0 ; i<TDSM_size ; i++ ; faire
6              a_preceded= e.activityId ; si TDSM[a_reference, a_preceded]>0 alors
7                  depFreq[a_reference, a_preceded]++ ;
8      /*dernière étape : construction de la table de dépendances finale */
9      pour int t_ref=0 ; t_ref <TDSM_size ; t_ref++ ; faire
10         pour int t_pr=0 ; t_pr <TDSM_size ; t_pr++ faire
11             TDSMF[t_ref, t_pr]= TDSM[t_ref, t_pr]/#t_ref ;
12 fin
```

Algorithme 3 : Calcul de la table de dépendances finale

Ainsi, notre algorithme adapte son comportement au contexte «concurrent». Cette stratégie permet l'amélioration de la complexité de l'exécution et du temps d'exécution de l'algorithme par rapport à une approche similaire de découverte de patrons fréquents [MTV97] qui utilise une taille fixe de la fenêtre de concurrence.

P(x,y)	STP	VSC	ER	MJR	AE	RP	RD	VD	LD
STP	1	0	0	0	0	0	0	-1	0
VSC	0	1	0	0	0	0	0	-1	0
ER	0	0	1	0	0	0	0	-1	0
MJR	0	1	0	0	0	0	0	-1	0
AE	0	1	0	0	0	0	0	-1	0
RP	0	1	0	0	0	0	0	-1	0
RD	0	0	0	0.58	0.23	0.19	0	-1	0
VD	1	-1	-1	-1	-1	-1	-1	0	0
LD	0	0	0	0	0	0	1	1	0

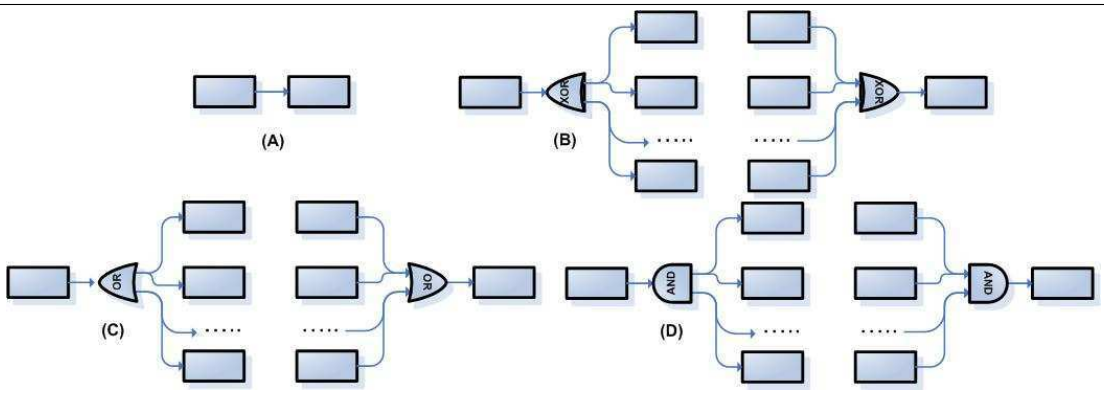
$$\#STP = \#VSC = \#VD = \#RD = \#LD = 100$$
$$\#ER = \#MJR = 69\,;\ \#AE = 21\,;\ \#RP = 10$$

TAB. 5.3 – Table Statistique de Dépendances finale ($P(x/y)$) et les fréquences des activités (#)

5.3.3 Propriétés statistiques des patrons de workflow

Nous identifions quatre cheminements possibles dans le `flot de contrôle` d'un workflow. D'une part, un cheminement séquentiel exclusif entre deux activités qui sont exclusivement interdépendantes (*c.f.* figure 5.7.a). D'autre part, on peut se trouver devant des opérateurs de «jointure» ou de «diffusion» où on doit faire un choix parmi l'ensemble de flux qui se trouve avant l'opérateur de «jointure» ou après le point de «diffusion». Nous avons identifié trois possibilités : un cheminement à choix exclusif où l'exécution choisit un seul flux (*c.f.* figure 5.7.b), un cheminement à choix libre qui ne pose pas de restriction sur le choix du nombre de flux à activer (*c.f.* figure 5.7.c) et un cheminement sans choix où l'exécution ne fait pas de choix et exécute tout les flux (*c.f.* figure 5.7.d).

Figure 5.7 Cheminements possibles des flux de contrôle



Ces cheminements dérivent trois sortes de comportements : séquentiel exclusif, conditionnel et concurrent. Nous avons décrit ces comportements en utilisant les propriétés statistiques extraites de la TDS finale. Ces propriétés vont être utilisées, par la suite, pour identifier séparément les patrons de workflow à partir des traces d'exécutions.

Nous commençons par le comportement de dépendance exclusif qui décrit un flux séquentiel unique. Le comportement de dépendance *mutuelle* exclusive entre deux activités spécifie que l'exécution d'une des deux activités dépend seulement de la fin de l'exécution de l'autre et que la fin d'exécution de la première activité déclenche seulement l'exécution de la deuxième. Le comportement de dépendance *mutuelle* exclusive est défini statistiquement dans la TDS finale par le corollaire 5.1.

COROLLAIRE 5.1 (PROPRIÉTÉS STATISTIQUES DU COMPORTEMENT *mutuelle* EXCLUSIVE)
Soit $\{A_k, 0 \leq k < n\}$ l'ensemble d'activités d'un workflow W. A_i et A_j deux activités de W. A_i et A_j décrivent un comportement de dépendance *mutuelle* exclusive de A_j vers A_i (noté *P1*) ***ssi*** en termes de :
- fréquences d'activités : $\#A_i = \#A_j$
- dépendances entre les activités : $P(A_i/A_j) = 1 \wedge \forall(0 \leq k, l < n;\ k \neq j; l \neq iP(A_i/A_k) = 0 \wedge P(A_l/A_j) = 0)$.

Le comportement parallèle lie un groupe d'activités en relation non-causale. Il spécifie comment, en termes de concurrence, l'exécution de ces activités est effectuée. Ce groupe d'activités se trouve dans le `flot de contrôle` après un opérateur de «diffusion» ou avant un opérateur de

«jointure». Nous avons distingué trois types de comportement parallèle :

1. **P2.1 : Concurrence globale** où dans la même instance de workflow tout le groupe d'activités est exécuté simultanément ;
2. **P2.2 : Concurrence partielle** où dans la même instance de workflow nous avons au moins une exécution concurrente partielle du groupe d'activités ;
3. **P2.3 : Pas de Concurrence** où il n'y a de comportement concurrentiel dans le groupe d'activités.

Le comportement parallèle sous ces trois formes est défini statistiquement dans la TDS finale par le corollaire 5.1.

COROLLAIRE 5.2 (PROPRIÉTÉS STATISTIQUES DU COMPORTEMENT PARALLÈLE)
Soit $\{A_i, 0 \leq i < n\}$ un groupe d'activités formant l'opérande se trouvant après un opérateur de «diffusion» ou avant un opérateur de «jointure». $\{A_i, 0 \leq i < n\}$ décrivent une :

1. **P2.1 : concurrence globale** ***ssi*** $\forall i, j; 0 \leq i \neq j < n; \#A_i = \#A_j \wedge P(A_i/A_j) = -1$
2. **P2.2 : concurrence partielle** ***ssi*** $\exists i, j; 0 \leq i \neq j < n\,; P(A_i/A_j) = -1$
3. **P2.3 : pas de concurrence** ***ssi*** $\forall i, j; 0 \leq i \neq j < n; \wedge P(A_i/A_j) \neq -1$

Le comportement conditionnel spécifie, en terme de `flot de contrôle`, comment s'effectue le choix d'activation parmi l'ensemble de flux qui se trouve après un opérateur de «diffusion» ou avant un opérateur de «jointure». Il définit une relation causale entre une activité et un groupe d'activités formant les opérandes des opérateurs de «diffusion» et de «jointure». Nous avons distingué trois types de comportement conditionnel :

- **P3.1 : Choix libre** où une partie du groupe d'activités est exécutée en fonction des contraintes et paramètres de chaque instantiation ;
- **P3.2 : choix unique** où seulement une activité est exécutée. Le choix de cette activité dépend des contraintes et paramètres de l'instance de workflow exécutée ;
- **P3.3 : pas de choix** où toutes les activités sont exécutées pour chaque instantiation.

Le comportement conditionnel, sous ces trois formes, est défini statistiquement dans TDS finale par le corollaire 5.3.

En utilisant les corollaires des propriétés statistiques 5.1, 5.2 et 5.3 qui dérivent tous du lemme 5.2, la dernière étape est la découverte des patrons de workflow en utilisant un ensemble de règles.

5.3.4 Règles de découverte des patrons de workflow

Chacun des patrons a ses propres règles statistiques qui abstraient statistiquement ses dépendances causales ou non-causales, et l'identifie d'une façon unique. Ces règles permettent, si les traces d'exécutions des workflows sont complètes, la découverte de tous les patrons de workflow inclus dans les traces d'exécutions. Elles procèdent par **une analyse locale des traces d'exécutions** qui permet en particulier **de récupérer des résultats partiaux** sous la forme de patrons. Essentiellement, pour découvrir un patron particulier de workflow nous avons besoin seulement d'événements concernant les éléments de ce patron. Ainsi, même en utilisant seulement des fractions de traces d'exécutions du workflow, nous pouvons découvrir correctement des patrons de workflow correspondant à ces traces d'exécutions (à condition que leurs événements appartiennent à ces fractions).

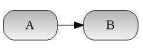
COROLLAIRE 5.3 (PROPRIÉTÉS STATISTIQUES DU COMPORTEMENT CONDITIONNEL)
Soit A une activité et $\{A_i, 0 \leq i < n\}$ un groupe d'activité formant les opérandes d'un opérateur de «diffusion» ou d'un «jointure». A et $\{A_i, 0 \leq i < n\}$ décrivent un :

- **P3.1 : Choix libre *ssi*** en termes de fréquences d'activités $(\#A \leq \Sigma_{i=0}^{n-1}(\#A_i)) \wedge (\#A_i \leq \#A)$ et en termes de dépendances entre les activités :
  - Dans un opérateur «diffusion» : $\forall 0 \leq i < n; P(A_i/A) = 1$ (A_i s'exécute certainement après l'occurrence de A)
  - Dans un opérateur «jointure» : $1 < \Sigma_{i=0}^{n-1} P(A/A_i) < n$ (A s'exécute certainement avant quelques occurrences des activités A_i d'où «1 <», mais pas toujours après tous les A_i d'où «$< n$»)
- **P3.2 : choix unique *ssi*** en termes de fréquences d'activités $(\#A = \Sigma_{i=0}^{n-1}(\#A_i)))$ et en termes de dépendances entre les activités :
  - Dans un opérateur «diffusion» : $\forall 0 \leq i < n; P(A_i/A) = 1$ (A_i s'exécute certainement après l'occurrence de A)
  - Dans un opérateur «jointure» : $\Sigma_{i=0}^{n-1} P(A/A_i) = 1$ (A s'exécute certainement avant l'occurrences d'une seule activités des A_i)
- **P3.3 : pas de choix *ssi*** en termes de fréquences d'activités $\forall i; 0 \leq i < n$, $\#A = \#A_i$ et en termes de dépendances entre les activités :
  - Dans un opérateur «diffusion» : $\forall 0 \leq i < n; P(A_i/A) = 1$ (A_i s'exécute certainement après l'occurrence de A)
  - Dans un opérateur «jointure» : $\forall 0 \leq i < n; P(A/A_i) = 1$ (A s'exécute certainement avant les occurrences de toutes les activités A_i)

Nous avons divisé les patrons de workflow en trois catégories (*c.f.* schéma 4.6) : le patron séquence, les patrons de «jointure» et de «diffusion». Dans la suite, nous présentons les règles qui nous ont permis de découvrir les patrons les plus communs et les plus utilisés appartenant à ces trois catégories.

5.3.4.1 Découvrir le patron «séquence» .

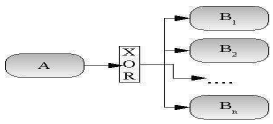
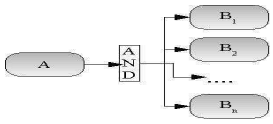
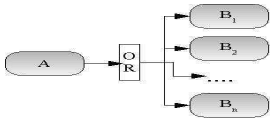
Dans cette catégorie, nous découvrons seulement le patron *séquence* (*c.f.* tableau 5.4). Dans ce patron formé par deux activités A et B, l'exécution de l'activité B dépend seulement de la terminaison de l'activité A. De ce fait, nous avons employé la propriété statistique de dépendance *mutuelle* exclusive (*c.f.* corollaire 5.1) pour découvrir cette relation liant B à A.

Rules	**patrons de workflow**
$(\#B = \#A)$	patron *séquence*
$(P(B/A) = 1)$	

TAB. 5.4 – Règle de découverte du patron *séquence*

Par exemple, en appliquant les règles de ce patron sur la TDS finale (*c.f.* tableau 5.3), nous découvrons le patron de séquence liant MJR et ER. En effet, $(\#ER = \#MJR)$ et $(P(MJR/ER) = 1)$ et $\forall A_{0 \leq i < n} \neq ER; P(MJR/A_i) = 0$ et $\forall A_{0 \leq j < n} \neq MJR; P(A_j/ER) = 0$.

5.3.4.2 Découvrir les patrons «diffusion»

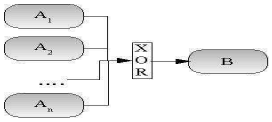
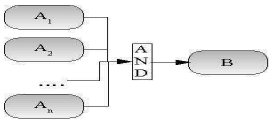
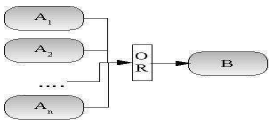
Règles	Patrons de workflow
$(\Sigma_{i=0}^{n}(\#B_i)=\#\mathrm{A})$	patron *choix exclusif*
$(\forall 0 \leq i \leq n; P(B_i/A)=1) \wedge$ $(\forall 0 \leq i,j \leq n;\ P(B_i/B_j)=0)$	
$(\forall 0 \leq i \leq n; \#B_i=\#\mathrm{A})$	patron *diffusion parallèle*
$(\forall 0 \leq i \leq n;\ P(B_i/A)=1) \wedge$ $(\forall 0 \leq i,j \leq n\ P(B_i/B_j)=-1)$	
$(\#A \leq \Sigma_{i=0}^{n}(\#B_i)) \wedge$ $(\forall 0 \leq i \leq n;\ \#B_i \leq \#A)$	patron *choix multiples*
$(\forall 0 \leq i \leq n;\ P(B_i/A)=1) \wedge$ $(\exists 0 \leq i,j \leq n; P(B_i/B_j)=-1)$	

TAB. 5.5 – Règles de découverte de patron de diffusion

Cette catégorie (*c.f.* tableau 5.5) a un opérateur «diffusion» où un flux de contrôle venant de l'activité A se divise en plusieurs flux de contrôle vers les activités $\{B_i; 0 \leq i \leq n\}$ qui peuvent être, selon le patron utilisé, exécutées ou pas.

La dépendance entre les activités A et $\{B_i; 0 \leq i \leq n\}$ avant et après l'opérateur «diffusion» diffère dans les trois patrons de cette catégorie : *choix exclusif*, *diffusion parallèle*, *choix multiples*. Ces dépendances sont décrites à travers, entre autres, les propriétés statistiques conditionnelles (*c.f.* corollaire 5.3). Le patron *choix exclusif*, où parmi plusieurs branches du flux du contrôle est choisie après l'opérateur de «diffusion», adopte la propriété de choix unique (P3.2)(*c.f.* corollaire 5.3). Quant aux patrons *diffusion parallèle* et *choix multiples*, ils se différencient par la propriété statistique du non-choix (P3.3) et la propriété (P3.1) du choix libre (*c.f.* corollaire 5.3). En fait, seulement une partie des activités est exécutée dans le patron *choix multiples* après un opérateur de «diffusion», alors que toutes les activités de B_i sont exécutées dans le patron *diffusion parallèle*. Le comportement de non-concurrence entre les activités B_i, dans le patron *choix exclusif*, est assuré par la propriété P2.3 (*c.f.* corollaire 5.2). Tandis que les comportement de la concurrence partielle et globale des patrons *diffusion parallèle*, *choix multiples* sont identifiés par l'application des propriétés statistiques P2.1 et P2.2 (*c.f.* corollaire 5.2).

Par exemple, la table TDS finale (*c.f.* tableau 5.3), indique que nous avons un patron `diffusion parallèle` liant STP, VSC et VD. En fait, il y a un parallélisme global entre VSC et VD ($P(VSC/VD) = -1 \wedge P(VD/VSC) = -1$) et ces activités sont toutes exécutées à la terminaison de STP ($P(VSC/STP) = 1 \wedge P(VD/STP) = 1$).

Règles	Patrons de workflow
$(\Sigma_{i=0}^{n}(\#A_i)=\#\mathrm{B})$	patron *jointure simple*
$(\Sigma_{i=0}^{n}\mathrm{P}(\mathrm{B}/A_i)=1) \wedge$ $(\forall 0 \leq i,j \leq n\,;\, P(A_i/A_j)=0)$	
$(\forall 0 \leq i \leq n;\, \#A_i=\#\mathrm{B})$	patron *synchronisation*
$(\forall 0 \leq i \leq n;\, P(B/A_i)=1) \wedge$ $(\forall 0 \leq i,j \leq n\; P(A_i/A_j)=-1)$	
$(m * \#B \leq \Sigma_{i=0}^{n}(\#A_i))$ $\wedge\ (\forall 0 \leq i \leq n;\, \#A_i \leq \#B)$ $(m \leq \Sigma_{i=0}^{n} P(B/A_i) \leq n)$ $\wedge\ (\exists 0 \leq i,j \leq n;\, P(A_i/A_j)=-1)$	patron *M-out-of-N-Join* 

TAB. 5.6 – Règles de découverte de patron de jointure

5.3.4.3 Découvrir les patrons «jointure».

Cette catégorie (*c.f.* tableau 5.6) a un opérateur de «jointure» où plusieurs flux de contrôle entrant fusionnent dans un unique flux de contrôle. Le nombre de branches nécessaires pour l'activation de l'activité B après opérateur «jointure» dépend du patron utilisé.

Pour identifier les trois patrons de cette catégorie : les patrons *jointure simple*, *synchronisation* et *M-out-of-N*, nous avons analysé les dépendances entre les activités A_i and B avant et après l'opérateur de «jointure». Ainsi, les propriétés du choix unique (P3.2) (*c.f.* corollaire 5.3) et de la non-concurrence (P2.3) (*c.f.* corollaire 5.2) sont utilisées pour identifier le patron de *jointure simple* où les flux de contrôle portant les activités A_i se rejoignent sans synchronisation ni concurrence. Les propriétés du non-choix (P3.3) (*c.f.* corollaire 5.3) et de concurrence globale (P2.3) (*c.f.* corollaire 5.2) sont toutes les deux utilisées pour identifier le patron *synchronisation* où toutes les activités A_i convergent ensemble en se synchronisant au point de «jointure». Au contraire du patron *M-out-of-N-Join*, où nous avons besoin seulement de la terminaison de M activités parmi les N flux de contrôle entrants, ainsi la concurrence entre les activités A_i est partielle (P2.2) (*c.f.* corollaire 5.2) et le choix est libre (P3.1) (*c.f.* corollaire 5.3).

Par exemple, en utilisant la table FSDT nous pouvons découvrir le patron *jointure simple* liant MJR, AE, RP et RD. En fait, les entrées de ces activités dans TDS finale indiquent un comportement non concurrent entre les activités MJR, AE et RP ($P(MJR/AE)$=$P(MJR/RP)$=$P(AE/RP)$ $\neq$ -1) et l'exécution de l'activité RD dépend de la terminaison exclusive d'une de ces trois activités MJR, AE et RP ($P(MJR/RD)$+$P(RP/RD)$+$P(RP/RD)$ = 1).

5.3.5 Composer les patrons de workflow découverts

Après la découverte des patrons de workflow, la construction du workflow final (complet) est faite par la composition des patrons de workflow découverts. En effet, dans notre approche

nous définissons le `flot de contrôle` d'un workflow comme l'union de patrons de workflow. Nous utilisons un système de réécriture illustré au tableau 5.7 pour lier les patrons de workflow découvert dans la section précédente et les combiner pour construire le workflow en entier.

Nous considérons une union de patrons comme un mot dont les terminaux sont les patrons. Ces terminaux peuvent être commutatifs dans le mot constituant le workflow. Notre système de réécriture nous permet de reconstruire le workflow en regroupant les patrons (les terminaux) par un ensemble de règles.

RR1 : *séquence*(a,b),$\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,b)$,$\{p_i\}$
RR2 : *diffusion parallèle*$(a,b_1,b_2,\ldots,b_n)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{B}(a,b_1,b_2,\ldots,b_n)$,$\{p_i\}$
R53 : *choix multiples*$(a,b_1,b_2,\ldots,b_n)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{C}(a,b_1,b_2,\ldots,b_n)$,$\{p_i\}$
RR4 : *choix exclusif*$(a,b_1,b_2,\ldots,b_n)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{D}(a,b_1,b_2,\ldots,b_n)$,$\{p_i\}$
RR5 : *synchronisation*$(a_1,a_2,\ldots,a_n,b)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{E}(a_1,a_2,\ldots,a_n,b)$,$\{p_i\}$
RR6 : *M-out-of-N*$(a_1,a_2,\ldots,a_n,b)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{F}\{a_1,a_2,\ldots,a_n,b)$,$\{p_i\}$
RR7 : *jointure simple*$(a_1,a_2,\ldots,a_n,b)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{G}(a_1,a_2,\ldots,a_n,b)$,$\{p_i\}$
RR8 : $\mathcal{A}(a,b)$, $\mathcal{A}(b,c)$,$\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$,$\{p_i\}$
RR9 : $\mathcal{A}(x,a)$, $\mathcal{F}sn(a,b_1,b_2,\ldots,b_n)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{F}sn(x,b_1,b_2,\ldots,b_n)$,$\{p_i\}$
RR10 : $\mathcal{F}sn(a,b_1,b_2,\ldots,b_n)$, $\mathcal{A}(b_i,x)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{F}sn(a_0,b_1,b_2,\ldots,b_{i-1},x,\ldots,b_n)$,$\{p_i\}$
RR11 : $\mathcal{A}(x,a_i)$, $\mathcal{J}nt(a_1,a_2,\ldots,a_n,b)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{J}nt(a_1,a_2,\ldots,a_{i-1},x,\ldots,a_n,b)$,$\{p_i\}$
RR12 : $\mathcal{J}nt(a_1,a_2,\ldots,a_n,b)$, $\mathcal{A}(b,x)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{J}nt(a_1,a_2,\ldots,a_n,x)\{p_i\}$
RR13 : $\mathcal{B}(a,b_1,b_2,\ldots,b_n)$, $\mathcal{E}(b_1,b_2,\ldots,b_n,c)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$, $\{p_i\}$
RR14 : $\mathcal{B}(a,b_1,b_2,\ldots,b_n)$, $\mathcal{F}(b_1,b_2,\ldots,b_n,c)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$, $\{p_i\}$
RR15 : $\mathcal{B}(a,b_1,b_2,\ldots,b_n)$, $\mathcal{G}(b_1,b_2,\ldots,b_n,c)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$, $\{p_i\}$
RR16 : $\mathcal{C}(a,b_1,b_2,\ldots,b_n)$, $\mathcal{F}(b_1,b_2,\ldots,b_n,c)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$, $\{p_i\}$
RR17 : $\mathcal{C}(a,b_1,b_2,\ldots,b_n)$, $\mathcal{G}(b_1,b_2,\ldots,b_n,c)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$, $\{p_i\}$
RR18 : $\mathcal{D}(a,b_1,b_2,\ldots,b_n)$, $\mathcal{G}(b_1,b_2,\ldots,b_n,c)$, $\{p_i\}$	$\longrightarrow$	$\mathcal{A}(a,c)$, $\{p_i\}$
RR19 : $\mathcal{A}(a,b)$, ε	$\longrightarrow$	$\mathcal{W}orkflow$

$\{\ p_i \in \mathcal{P}atron | p_i \in$ workflow découvert $\}$

$\mathcal{F}sn = \mathcal{B} \vee \mathcal{C} \vee \mathcal{D}$

$\mathcal{J}nt = \mathcal{E} \vee \mathcal{F} \vee \mathcal{G}$

TAB. 5.7 – Règles de réécriture définissant une grammaire de toutes les combinaisons cohérentes des patrons de workflow découverts

Cet ensemble de règles nous assure de découvrir un modèle de workflow orienté patron sûr et bien formé. Par conséquent, en utilisant ces règles de réécriture nous sommes sûrs que les patrons de workflow découverts ne contiennent aucun flux erronés. Afin de ne pas avoir des unions non sensées (patrons disjoints) ou incohérentes (avec des sémantiques incompatibles), la grammaire de ce système de réécriture définit un langage d'unions cohérentes. Ainsi, un `flot de contrôle` découvert est cohérent ***ssi*** l'union des patrons de workflow découverts correspondants est un mot généré par cette grammaire. Concrètement cette grammaire postule que :

- Un `flot de contrôle` doit commencer par un des patrons *séquence*, *diffusion parallèle*, *choix multiples* ou *choix exclusif* (les règles de réécriture RR8, RR13, RR14, RR15, RR16, RR17, RR18, RR19).
- Tout patron peut être suivi ou précédé par un patron *séquence* (les règles de réécriture RR8, RR9, RR10, RR11, RR12).
- un patron *diffusion parallèle* doit être suivi par un des patrons *synchronisation*, *M-out-of-N*

ou *jointure simple* ou *séquence* (les règles de réécriture RR10, RR13, RR14, RR15).
- un patron *choix multiples* doit être suivi par un des patrons *M-out-of-N* ou *jointure simple* ou *séquence* (les règles de réécriture RR10, RR16, RR17).
- un patron *choix exclusif* doit être uniquement suivi par un patron *jointure simple* ou un patron *séquence* (la règles de réécriture RR10, RR18).

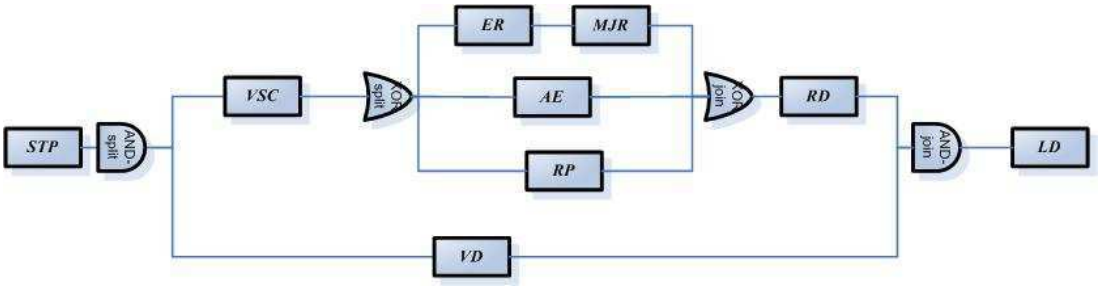
Exemple : En appliquant les règles de découverte sur la TDS finale (*c.f.* tableau 5.3) de notre workflow de prêt bancaire illustré par la figure 5.8, nous trouvons que le `flot de contrôle` découvert est défini comme l'union de ces patrons :

diffusion parallèle(STP,VSC,VD), *choix exclusif*(VSC, ER, AE, RP), *jointure simple*(MJR, AE, RP, RD), *séquence*(ER, MJR), *synchronisation*(VD,RD,LD)

En appliquant les règles de réécriture du tableau 5.7 sur ce mot, nous pouvons combiner ces patrons de workflow découverts, en les liant dans une structure cohérente pour reconstruire, notre workflow de prêt bancaire :

diffusion parallèle(STP,VSC,VD), *choix exclusif*(VSC, ER, AE, RP), *jointure simple*(MJR, AE, RP, RD), *séquence*(ER, MJR), *synchronisation*(VD,RD,LD) $\longrightarrow_{RR1,RR2,RR3,RR4,RR5,RR6,RR7}$ $\mathcal{B}$(STP,VSC,VD), $\mathcal{D}$(VSC, ER, AE, RP), $\mathcal{G}$(MJR, AE, RP, RD), $\mathcal{A}$(ER, MJR), $\mathcal{E}$(VD,RD,LD) $\longrightarrow_{RR11}$ $\mathcal{B}$(STP,VSC,VD), $\mathcal{D}$(VSC, ER, AE, RP), $\mathcal{G}$(ER, AE, RP, RD), $\mathcal{E}$(VD,RD,LD) $\longrightarrow_{RR18}$ $\mathcal{B}$(STP,VSC,VD), $\mathcal{A}$(VSC, RD), $\mathcal{E}$(VD,RD,LD) $\longrightarrow_{RR10}$ $\mathcal{B}$(STP,VD,RD), $\mathcal{E}$(VD,RD,LD) $\longrightarrow_{RR13}$ $\mathcal{A}$(STP,LD) $\longrightarrow_{RR19}$ $\mathcal{W}orkflow$

Figure 5.8 Le `flot de contrôle` découvert du workflow de prêt bancaire



5.3.6 Synthèse

Dans cette section, nous avons spécifié des techniques d'analyse statistique des traces d'exécutions pour la découverte du `flot de contrôle`. Notre analyse statistique nous a permis de construire une table de dépendances statistiques entre les activités. Cette table a été utilisée par la suite comme base pour la découverte des patrons de workflow formant le `flot de contrôle` à travers un ensemble de règles qui isolent statistiquement les propriétés comportementales des patrons.

Notre contribution pour la découverte du `flot de contrôle` propose une nouvelle approche caractérisée par une **découverte locale** des patrons de workflow spécifiant un modèle orientés blocs bien structuré et correcte. En effet, même si nous avons seulement que des fractions de traces d'exécutions, nous pouvons découvrir des résultats partiels sous la condition que ces fractions respectent la condition de complétude de traces d'exécutions des activités qu'elles contiennent. Concrètement, dans le cas où les traces d'exécutions ne contiennent que les traces d'une partie des activités du workflow, nous pouvons grâce à notre approche de **découverte locale** des patrons de workflow extraire les patrons relatifs à ces activités. Au contraire de la plupart des techniques

antérieures de découverte de workflow qui découvrent le workflow en entier, ce qui nécessite la présence des traces d'exécutions de toutes les activités du workflow.

Par ailleurs, la découverte des patrons de workflow permet de découvrir des comportement **plus complexes** avec une meilleure spécification de l'opérateur «diffusion» (les patrons *diffusion parallèle*, *choix exclusif*, *choix exclusif*) et de l'opérateur «jointure» (les patrons *synchronisation*, *M-out-of-N-Join*, and *M-out-of-N-Join*). Elle traite mieux le problème de concurrence à travers l'introduction du concept de la «fenêtre de concurrence» qui traite **dynamiquement** la concurrence. En effet, la dynamicité de notre algorithme réside dans le fait que la taille de la «fenêtre de concurrence» n'est pas statique ou fixe, au contraire elle est variable d'une activité à une autre selon son comportmemet concurrentiel. Bien que l'algorithme décrit dans notre approche semble être plus simple dans le calcul, cette simplicité n'affecte pas son efficacité dans le traitement de l'aspect concurrentiel des workflows.

En conclusion, cette section nous a permis de traiter de la troisième interrogations de notre problématique : « Comment les techniques de fouille de données peuvent-elles être utilisées pour la découverte des procédés ?» en proposant des techniques d'analyse statistique des traces d'exécutions et en adoptant des techniques de découverte de la concurrence dans les travaux de découverte de patrons fréquents dans «un chaînage de phrases» [MTV97].

Le tableau 5.8 compare notre approche de découverte du `flot de contrôle` à d'autres approches existantes et validées par des outils. Nous nous focalisons sur dix aspects :

- La **structure** du langage cible découvert : se rapporte à la quantité et à la qualité de l'information qui est directement montrée dans la structure du modèle de procédés découvert. Quelques techniques découvrent un modèle de procédés qui exprime seulement les dépendances entre activités ; d'autres expriment également la sémantique des points de « jointure» ou de «diffusion» par des réseaux de petri, etc. En plus, quelques techniques de découverte visent à extraire le modèle entier, d'autres se concentrent seulement sur l'identification de sous-structures les plus communes dans le modèle de procédé.
- La **découverte locale** : indique si la technique de découverte adoptée peut traiter des fragments de traces d'exécutions ou si elle procède par une approche qui découvre le workflow en entier ce qui exclut un traitement partiel de fragments de traces d'exécutions et nécessite les traces d'exécutions de toutes les activités.
- Les **traces d'exécutions** : indiquent si la technique suppose que les traces d'exécutions ont des informations sur le début et la fin d'exécution d'une activité (activité non-atomique) ou pas (activité atomique).
- L'**approche de découverte** : indique si la technique essaye de découvrir le modèle de procédés par une étape unique ou si l'approche a des étapes multiples avec des structures intermédiaires qui sont raffinés dans ces étapes.
- Le **parallélisme** : indique si la technique supporte des flux de contrôle concurrentiels.
- Le **choix non-libre** : vérifie si l'approche peut supporter la structure de contrôle du choix non libre qui est un mélange à la fois de la synchronisation et du choix dans un seul opérateur.
- Les **boucles** : vérifient si l'approche peut supporter les transitions cycliques.
- Les **boucles courtes** : boucles se composant d'une transition cyclique de deux activités.
- Le **bruit** : situation où les traces d'exécutions contiennent des erreurs ou des exemples qui peuvent pas être représentés par le langage.
- Le **temps** : vérifie si l'approche nécessite une structure de traces d'exécutions enrichies (qui contient par exemple une information sur le temps d'exécution calculé pour les indicateurs de performance d'exécution tels que des temps d'attente/synchronisation, des temps d'écoulement, le taux de chargement/utilisation, etc.), dans le cas contraire l'approche se

satisfait d'une structure de traces d'exécutions simplifiée comme nous l'avons décrit dans la définition 5.3.

outils de découverte	EMiT [vdAvD02]	Little Thumb [WvdA02]	InWoLvE [HK04]	Process Miner [Sch02]	WorkflowMiner [GBG05; BGKM06]
Structure	Petri Nets	Graphe	Dépendance	Blocs	**Patrons**
Découverte locale	Non	Non	Non	Non	**Oui**
traces d'exécutions	Atomique	Atomique	Atomique	Non atomique	**Atomique**
Parallélisme	Oui	Oui	Oui	Oui	**Oui**
Approche	plusieurs	unique	unique	plusieurs	**plusieurs**
Choix non-libre	Non	Non	Non	Non	**Oui**
Boucles	Oui	Oui	Non	Non	**Oui**
Boucles courtes	Oui	Oui	Non	Non	**Non**
Bruit	Non	Oui	Oui	Non	**Non**
Temps	Oui	Non	Non	Non	**Non**

TAB. 5.8 – Comparaison des approches pour la découverte du `flot de contrôle`

En conclusion, notre approche répond d'une façon optimale à toutes les propriétés définies dans ce tableau sauf les caractéristiques de boucle courte et de bruit. Notons que la découverte des boucles courtes (2 activités) et le traitement du bruit ont été respectivement traités dans les travaux de [WvdA02; dMvDvdAW04] et [vdAvD02] qui proposent une étape séparée de prétraitement des traces d'exécutions. Nous avons choisi de ne pas inclure cette étape dans notre approche dans le souci de ne pas alourdir le traitement dans notre algorithme. Nous pensons actuellement à une méthode plus originale et moins lourde pour répondre à ces deux points.

5.4 Découverte du `flot transactionnel`

Nous détaillons, dans cette section, notre approche pour la découverte et l'amélioration des techniques de recouvrement de workflows. Notre défi est d'exploiter les traces d'exécutions enrichies par le cycle de vie des activités pour découvrir le comportement transactionnel du workflow. Ces informations supplémentaires sont nécessaires à nos techniques de découverte du `flot transactionnel`. Des traces d'exécutions ne contenant pas d'informations sur le cycle de vie des activités sont insuffisantes pour procéder à la découverte du `flot transactionnel`. Ainsi, nous adoptons dans ce qui suit le modèle de traces d'exécutions «enrichi» introduit dans la figure 5.3 et contenant les différents états de l'activité.

Comme nous avons fait pour découvrir les patrons de workflow, nous construisons des tables statistiques de dépendances transactionnelles qui informent seulement sur les dépendances d'événements capturées après des échecs d'activités. Pour calculer ces dépendances, nous utilisons la même définition (utilisée dans section 5.3.2), sauf que nous capturons seulement des dépendances d'événements après des échecs d'activités. Concrètement, chaque table statistique de dépendances transactionnelles est liée à une activité *act* et capture statistiquement le comportement du workflow après les échecs de *act*. Nous distinguons deux types de dépendances transactionnelles : (i) des dépendances statistiques transactionnelles inter-activités (*c.f.* section 5.4.1) pour découvrir les dépendances transactionnelles d'alternative, de suspension, et d'annulation, et (ii) des dépendances statistiques transactionnelles intra-activité (*c.f.* section 5.4.2) pour découvrir les propriétés transactionnelles.

5.4.1 Découverte des dépendances transactionnelles

5.4.1.1 Construction de la table de dépendances transactionnelles inter-activités

Dans le chapitre précédent (*c.f.* section 4.4) après un échec d'exécution d'une activité (c.à.d l'exécution de l'action `échouer()`), le workflow se trouve dans un état incohérent. Des actions inter-activités, spécifiées par des dépendances d'exécutions exceptionnelles (*c.f.* définition 4.7) sont mises alors à l'œuvre par les entités externes (programmateur, intervention humaine, etc.) et permettent à l'activité échouée d'agir avec l'extérieur pour retrouver un état cohérent. Le but est de remettre le workflow échoué de nouveau dans un état sémantiquement acceptable. Ainsi, l'état incohérent d'échec peut alors être corrigé et l'exécution peut reprendre grâce au mécanisme de recouvrement qui ramène le workflow à un point cohérent avec l'espoir qu'il pourra se terminer alors avec succès.

Nous avons décrit, dans le chapitre précédent, différents types de dépendances transactionnelles qui décrivent le comportement d'une activité et ses interactions avec les autres activités en cas d'échec. Ces dépendances transactionnelles permettent de décrire les mécanismes de recouvrement nécessaires à la poursuite d'exécution de l'instance de workflow. Ces différents types de dépendances transactionnelles inter-activités peuvent être découvertes à travers les statistiques des table de dépendances transactionnelles inter-activités ITR. En effet, si une activité act appartenant au workflow échoue, une table ITR_{act} est construite pour capturer ses interactions avec l'extérieur après l'échec. Notons que ses interactions avant l'échec ne sont pas relatées dans la table ITR_{act}.

Pour la construction de ITR_{act}, nous ne traitons que les traces d'exécutions des instances où act échoue ne gardant que les dépendances d'événements après l'échec de act. Les dépendances se trouvant avant l'échec de act concernent le `flot de contrôle`. Nous distinguons deux types de dépendances transactionnelles inter-activités dans ITR_{act} (*c.f.* définition 5.6). La première catégorie concerne évidemment les dépendance d'événements dans lesquels on trouve l'événement d'échec de l'activité act (e.state= `échoué` $\wedge$ e.activity = act). La deuxième catégorie concerne toute dépendance d'événement où il y a une activité exécutée après l'échec de act. Dans cet ensemble d'activités exécutées après l'échec de act, on peut trouver des activités qui ne se trouvent pas dans le `flot de contrôle` découvert. En effet, apporter le workflows de nouveau à un état sémantiquement acceptable peut également nécessiter de compenser l'échec de act par des «nouvelles» activités de compensation qui défont sémantiquement act [KMO98b]. Ces activités de compensation n'existent pas initialement dans le `flot de contrôle` découvert.

Définition 5.6 (Statistiques de dépendances transactionnelles inter-activités)
Nous dénotons par ITR_{act} la table de dépendances transactionnelles inter-activités qui rapporte les dépendances d'événements après l'échec de act. Chaque entrée $ITR_{act}(e_1, e_2)$ dans la table est une dépendance d'événement où pour i=1 ou i=2 :
- (e_i.state= `échoué` $\wedge$ e_i.activity = act ;) $\vee$
- $\exists str : EventStream; Evti, Evtj : Event \in str |$ ($Evtj$.activity = act $\wedge$ $Evtj$.state= `échoué` $\wedge$ $Evti$.activity = e_i.activity $\wedge$ $Evti$.state= e_i.state $\wedge$ $Evtj$.occurTime > $Evti$.occurTime) ;

La table 5.9 capture une fraction de la table statistique des dépendances transactionnelles inter-activités après l'échec de l'activité RD (ITR_{RD}) . Nous pouvons constater que dans cette table il y a une nouvelle activité CR qui n'existe pas dans le `flot de contrôle` découvert. Cette activité est exécutée pour compenser les échecs de RD .

ITR_{RD}	RD,e	CR,a	RE,a	AE,a	RP,a	MJR,a
RD,e	0	0	0	0.56	0.34	0.1
CR,a	1	0	0	0	0	0
ER,a	0	1				
AE,a	0	1				
RP,a	0	1				
MJR,a	0	0				

i=`initial`, a=`activé`, s=`suspendu` t=`terminé`, e=`échoué`, n=`annulé`

TAB. 5.9 – Fractions de la table de dépendances statistiques transactionnelles inter-activités de l'activité RD

5.4.1.2 Spécification statistique des dépendances transactionnelles

Après l'échec d'une activité A_i, la procédure de recouvrement peut être initialisée par une dépendance alternative. Le corollaire 5.4 nous permet de découvrir les dépendances alternatives d'une activité A_i à partir de la table ITR_{A_i}. Ces dépendances sont constatées si on observe une entrée non nulle entre l'événement rapportant l'état `échoué` de A_i et l'événement rapportant l'état `activé` d'une autre activité A_j dans ITR_{A_i}. Selon la localisation de A_j, nous pouvons identifier deux types de dépendances alternatives : une alternative en avant (respectivement en arrière) si A_j se trouve avant (respectivement après) A_i dans le `flot de contrôle` découvert. Dans le cas où A_j est une activité qui ne se trouve pas dans le `flot de contrôle` découvert alors on observe une alternative en avant (respectivement en arrière) s'il existe une activité A_k dans le `flot de contrôle` découvert tel que A_j est exécutée avant A_k dans ITR_{A_i} et A_k se trouve avant (respectivement après) A_i dans le `flot de contrôle` découvert.

COROLLAIRE 5.4 (PROPRIÉTÉS STATISTIQUES D'UNE DÉPENDANCE ALTERNATIVE)
Il y a une dépendance alternative de A_i à A_j ***ssi*** l'échec de A_i peut causer l'activation de A_j. En utilisant les dépendances statistiques transactionnelles de la table ITR_{A_i}, nous avons :

$$depAlt(A_i, A_j) \Longleftrightarrow ITR_{A_i}\ ((A_j, \texttt{activé}), (A_i, \texttt{échoué})) \neq 0.$$

Nous avons distingué deux types de dépendances alternatives :
- alternative en arrière : si le point cohérent se trouve avant l'activité échouée. Exprimé statistiquement $A_j \lhd_{TDS} A_i \vee \exists A_k |\ A_k \lhd_{TDS} A_i \wedge A_j \lhd_{ITR} A_k$
- alternative en avant : si le point cohérent se trouve après l'activité échouée. Exprimé statistiquement $A_i \lhd_{TDS} A_j \vee \exists A_k |\ A_i \lhd_{TDS} A_k \wedge A_j \lhd_{ITR} A_k$

Exemple : Dans le workflow exemple de prêt bancaire, nous pouvons déduire de la table 5.9 que nous avons une dépendance alternative en arrière de RD vers une nouvelle activité CR. En effet, $ITR_{RD}\ ((CR, \texttt{activé}), (RD, \texttt{échoué})) = 1 \wedge RD \lhd_{TDS} ER \wedge ER \lhd_{ITR} CR$

En plus, un échec d'une activité peut causer une annulation (fin non régulière et anormale) d'une ou plusieurs activités actives. Le corollaire 5.5 nous permet de découvrir les dépendances d'annulation d'une activité A_i à partir de la table ITR_{A_i}. Ces dépendances sont constatées si on observe une entrée non nulle entre l'événement rapportant l'état `échoué` de A_i et l'événement rapportant l'état `annulé` d'une autre activité A_j dans ITR_{A_i}.

Finalement, un échec d'une activité peut causer une suspension temporaire d'exécution d'une

COROLLAIRE 5.5 (PROPRIÉTÉS STATISTIQUES D'UNE DÉPENDANCE D'ANNULATION)
Il y a une dépendance d'annulation de A_i à A_j ***ssi*** l'échec de A_i peut causer l'annulation de A_j. En utilisant les dépendances statistiques transactionnelles de la table ITR_{A_i}, nous avons :

$$depAnl(A_i, A_j) \Longleftrightarrow ITR_{A_i}((A_j,\texttt{annulé}),\ (A_1,\texttt{échoué})\ ) \neq 0$$

ou plusieurs activités actives. L'exécution de ces activités pourrait reprendre après la fin du mécanisme de recouvrement. Le corollaire 5.5 nous permet de découvrir les dépendances de suspension d'une activité A_i à partir de la table ITR_{A_i}. Ces dépendances sont constatées si on observe une entrée non nulle entre l'événement rapportant l'état `échoué` de A_i et l'événement rapportant l'état `suspendu` d'une autre activité A_j dans ITR_{A_i}.

COROLLAIRE 5.6 (PROPRIÉTÉS STATISTIQUES D'UNE DÉPENDANCE DE SUSPENSION)
Il y a une dépendance de suspension de A_i à A_j ***ssi*** l'échec de A_i peut causer la suspension A_j. En utilisant les dépendance statistiques transactionnelles de la table ITR_{A_i}, nous avons :

$$depSus(A_i, A_j) \Longleftrightarrow ITR_{A_i}((A_j,\texttt{suspendu}),\ (A_i,\texttt{échoué})\ ) \neq 0$$

5.4.2 Découverte des propriétés transactionnelles

5.4.2.1 Construction de la table de dépendances transactionnelles intra-activité

Comme nous l'avons énoncé dans le chapitre précédent (*c.f.* section 4.4.1.1), un diagramme de cycle de vie peut être associé à chaque activité qui modélise les états possibles par lesquels les exécutions de cette activité peuvent passer, et les transitions possibles entre ces états. Ces transitions dépendent de ces propriétés transactionnelles. Nous avons spécifié trois propriétés transactionnelles : **rejouable**, **atomique** et **pivot** qui sont caractérisées par un cycle de vie unique.

DÉFINITION 5.7 (STATISTIQUES DE DÉPENDANCES TRANSACTIONNELLES INTRA-ACTIVITÉ)
Nous dénotons par ATR^A_{act} la table de dépendances statistiques transactionnelles intra-activité qui rapporte les dépendances internes des événements de l'activité A après l'échec de *act*. Ces transitions sont extraites à partir d'une projection des traces d'exécutions d'instances où *act* échoue prenant seulement les événements de A. Chaque entrée $ATR^A_{act}(e_1, e_2)$ dans la table représente une dépendance des événements internes de A où pour i=1, et i=2 :
– $(e_i.\text{activity} = A\ ;\ ) \wedge$
– $\exists str : EventStream; Evti, Evtj : Event \in str |\ (Evtj.\text{activity} = act \wedge Evtj.\text{state}= \texttt{échoué} \wedge Evti.\text{activity} = A \wedge Evti.\text{state}= e_i.\text{state} \wedge Evtj.\text{occurTime}>Evti.\text{occurTime})$;

Les différents types de propriétés transactionnelles sont découvertes à travers les statistiques des table ATR (*c.f.* définition 5.7). En effet, si une activité *act* appartenant au workflow échoue, une table ATR^A_{act} est construite pour chaque activité A se trouvant après *act* pour capturer les dépendances internes entre les événements de A se produisant après l'échec de *act*. Ainsi la table

ATR_{act} extrait des traces d'exécutions des instances où *act* échoue les dépendances entre les événements internes des activités qui la suivent.

La table 5.10 capture les tables statistiques des dépendances transactionnelles intra-activité des activités ER, AE, RP et CR après l'échec de l'activité RD ($ATR^{ER}_{RD}, ATR^{AE}_{RD}, ATR^{RP}_{RD}, ATR^{CR}_{RD}$). Par exemple, la table ATR^{ER}_{SD} indique que l'activité de ER est réactivée après l'échec de RD. En particulier, on remarque qu'elle pourrait être initialisée à nouveau pour être re-exécutée après l'échec de RD bien qu'elle ait atteint l'état terminé (ATR^{ER}_{RD}(`terminé`, `initial`)= 0.22).

ATR^{ER}_{RD}	i	a	s	n	e	t
i				0	0	0.22
a	1					
s		0				
t		1				
n		0				
e		0	0			

ATR^{RP}_{RD}	i	a	s	n	e	t
i				0	0	0.24
a	1					
s		0				
t		1				
n		0				
e		0	0			

ATR^{AE}_{RD}	i	a	s	n	e	t
i				0	0	0.54
a	1					
s		0				
t		1				
n		0				
e		0	0			

ATR^{CR}_{RD}	i	a	s	n	e	t
i						
a	1					
s		0				
t		1				
n		0				
e		0	0			

i=`initial`, a=`activé`, s=`suspendu` t=`terminé`, e=`échoué`, n=`annulé`

TAB. 5.10 – Fractions de la table de dépendances statistiques transactionnelles intra-activité de l'activité RD

5.4.2.2 Spécification statistique des propriétés transactionnelles

Une activité a est dite **rejouable** (a^r) si elle se termine toujours avec succès après un nombre fini d'activations. Le corollaire 5.7 nous permet de découvrir la propriété transactionnelle **rejouable** d'une activité a à partir des tables ATR^a et ITR_a. Cette propriété est vérifiée si on observe dans la table ATR^a_a une dépendance égale à 1 entre l'événement rapportant l'état `échoué` de a et l'événement rapportant l'état `activé` de la même activité a et on constate aussi dans la table ITR_a une entrée égale à 1 entre l'événement rapportant l'état `échoué` de a et l'événement rapportant l'état `activé` de la même activité a.

COROLLAIRE 5.7 (PROPRIÉTÉS STATISTIQUES DE LA PROPRIÉTÉ **REJOUABLE**)
Une activité a est dite rejouable (a^r) ***ssi*** son échec entraîne sa re-exécution, jusqu'elle atteint l'état `terminé`.
En utilisant les dépendances statistiques transactionnelles des tables ATR^{a^r} et ITR_{a^r}, nous avons :

$$a^r \iff ATR^{a^r}_{a^r}(\texttt{activé}, \texttt{échoué})=1 \wedge ITR_{a^r}((a^r, \texttt{activé}), (a^r, \texttt{échoué})) = 1$$

Une activité a est dite **pivot** (a^p) si une fois qu'elle se termine avec succès, elle ne peut être re-exécutée et ainsi les effets de son exécution sont persistants. Le corollaire 5.8 nous permet de découvrir la propriété transactionnelle **pivot** d'une activité a à partir des tables ATR^a. Cette propriété est vérifiée si on observe qu'aucune des tables ATR^a ne rapporte une entrée non nulle entre les événements rapportant les états `terminé`, `échoué` ou `annulé` d'une part et l'événement rapportant l'état `activé` de la même activité a^p, d'autre part.

COROLLAIRE 5.8 (PROPRIÉTÉS STATISTIQUES DE LA PROPRIÉTÉ **PIVOT**)
Une activité a est dit pivot (a^p) ***ssi*** l'échec d'une autre activité ne peut modifier l'état final qu'elle a atteint à la fin de son premier et unique cycle de vie en la re-initialisant.
En utilisant les dépendance statistiques transactionnelles des tables ATR^{a^p}, nous avons :

$$a^p \Longleftrightarrow \nexists\, act | (\, act \neq a^p \wedge ATR^{a^p}_{act}(\text{«}x\text{»}, \texttt{initial}) \neq 0 \wedge (\text{«}x\text{»} = \texttt{terminé} \vee \text{«}x\text{»} = \texttt{échoué} \vee \text{«}x\text{»} = \texttt{annulé})).$$

Exemple : Dans le workflow exemple de prêt bancaire, nous pouvons déduire des tables 5.10 que les activités ER, MJR, AE et RP ne sont pas pivots. En effet, $ATR^{ER}_{RD}(\texttt{échoué}, \texttt{initial}) = ATR^{MJR}_{RD}(\texttt{terminé}, \texttt{initial}) = ATR^{RP}_{RD}(\texttt{terminé}, \texttt{initial}) = ATR^{AE}_{RD}(\texttt{terminé}, \texttt{initial}) \neq 0$ indique que ces activités ne gardent pas l'état `terminé` et sont re-initialisées à nouveau pour être re-exécuté après l'échec de RD. Ainsi l'état `terminé` n'est pas persistant.

Une activité a est dite **atomique** (a^a) si elle ne peut pas être ni annulée ni suspendue définitivement et qu'elle est sûre de se terminer avec succès après son activation. Le corollaire 5.9 nous permet de découvrir la propriété transactionnelle **atomique** d'une activité a à partir des tables ATR^a et ITR_a. Cette propriété est vérifiée si on observe que la table ITR_a est vide (l'activité n'échoue jamais) et qu'on constate qu'aucune des tables ATR^a ne rapporte une entrée non nulle entre les événements rapportant l'état `activé`, et l'état `annulé` de a^a, et que si l'activité est `suspendue` après l'échec d'une autre activité elle est sûre que cet état est non persistant puisqu'elle sera re-activée.

COROLLAIRE 5.9 (PROPRIÉTÉS STATISTIQUES DE LA PROPRIÉTÉ **ATOMIQUE**)
Une activité a est dite atomique (a^a) *ssi* elle ne peut pas échouer ou être suspendue ou annulée après l'échec d'une autre activité
En utilisant les dépendances statistiques transactionnelles des tables ATR^{a^a} et ITR_{a^a} nous avons :

$$a^a \Longleftrightarrow (ITR_{a^a} = \emptyset) \wedge (\nexists\, act |\, (act \neq a^a \wedge ATR^{a^a}_{act}(\texttt{annulé}, \texttt{activé}) = 0)) \wedge (ATR^{a^a}_{act}(\texttt{suspendu}, \texttt{activé}) > 0 \Rightarrow ATR^{a^a}_{act}(\texttt{activé}, \texttt{suspendu}) = 1)$$

5.4.3 Synthèse

Dans cette section, nous nous sommes intéressés à la découverte du comportement transactionnel du workflow. Pour ce faire, Nous avons proposé des techniques d'analyse statistique des traces d'exécutions pour la découverte du `flot transactionnel` et des propriétés transactionnelles des activités. Notre analyse statistique porte exclusivement sur les instances de workflow

contenant des échecs d'exécution. En plus, la découverte du comportement transactionnel nécessite une structure «enrichie» de traces d'exécutions par rapport à celle utilisée pour la découverte du `flot de contrôle`. Ces traces d'exécutions doivent contenir, en particulier, l'information portant les différents états des activités lors d'une instance d'exécution décrivant ainsi leurs cycles de vie.

Les techniques d'analyse statistique, que nous avons utilisées pour la découverte du `flot transactionnel` sont assez similaires à celles employées dans la section 5.3 pour la découverte du `flot de contrôle`. En effet, nous procédons par la construction de tables de dépendances statistiques transactionnelles, sauf que ces tables capturent, non pas les dépendances entre les activités, mais les dépendances entre les événements des activités. Nous y distinguons deux types de table : la table de dépendances statistiques transactionnelles intra-activité que nous utilisons pour la découverte des propriétés transactionnelles des activités et la la table de dépendances statistiques transactionnelles inter-activités que nous utilisons pour la découverte du `flot transactionnel` à travers un ensemble de corollaires qui dérivent tous du lemme 5.2.

En conclusion, cette section nous a permis de traiter de la quatrième interrogation de notre problématique : «Quel outil utiliser pour l'analyse des mécanismes de recouvrement et l'amélioration de la fiabilité d'exécution du procédé ?» en proposant d'analyser les traces d'exécutions contenant des échecs d'exécution pour découvrir le comportement transactionnel contenant les mécanismes de recouvrement.

5.5 Re-ingénierie du comportement transactionnel

Dans cette section, nous proposons des techniques d'amélioration et de correction, dans le cadre d'une re-ingénierie des procédés métiers. Notre but est de fournir un outil d'aide à une conception correcte, proche des choix métiers et conceptuels du concepteur, et donnant lieu à une exécution fiable. Nous nous intéressons, en particulier, à la correction et l'amélioration du comportement transactionnel.

Concrètement, nous allons utiliser la découverte de procédé pour une Delta analyse (*c.f.* section 5.5.1), c.à.d, comparer le procédé réel, représenté par le workflow découvert, au procédé prédéfini initialement conçu. En comparant le modèle de procédé initialement conçu au modèle découvert, des disparités entre les deux modèles peuvent être détectées et utilisées pour améliorer, en particulier, le comportement transactionnel du procédé. Pour ce faire, nous proposons, par la suite, un ensemble de règles qui permettent de corriger ou de supprimer, si nécessaire, tout comportement transactionnel erroné ou inutile et de réduire au minimum la quantité d'efforts reprenant l'exécution de workflows en cas d'échec d'exécution d'une activité. Par comportements transactionnels erronés ou inutiles, nous voulons dire le `flot transactionnel` initialement spécifiés qui n'est pas nécessaire ou qui ne coïncide pas avec la réalité d'exécution exprimant des besoins d'évolution des concepteurs ou des erreurs de choix conceptuels faites à la première phase de conception. En effet, ces comportements transactionnels peuvent être simplement coûteux mais également source d'erreur.

Les règles de correction et d'amélioration, que nous spécifions, dépendent de la sémantique transactionnelle du procédé. En effet, les spécifications de comportement transactionnel de workflows doivent respecter les choix métiers et conceptuels du concepteur et les règles sémantiques présentées dans le chapitre précédent (*c.f.* règle 4.1). Ainsi, nous procédons par un ensemble de règles qui nous permettent de :

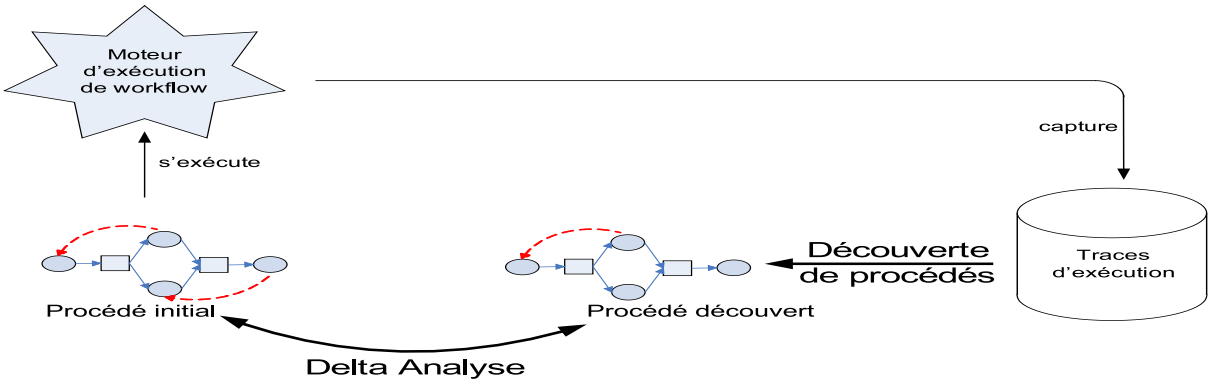
- Corriger ou enlever tout comportement transactionnel découvert erroné (*c.f.* section 5.5.2),
- Suggérer un ensemble de comportements transactionnels additionnels (*c.f.* section 5.5.3).

Nous notons $a_{initial}$ le comportement transactionnel de l'activité a à la phase initiale de conception. $a_{decouvert}$ désigne le comportement transactionnel découvert de l'activité a. Et, $a_{corrige}$ désigne le comportement transactionnel corrigé ou amélioré de l'activité a.

5.5.1 Delta analyse

Dans la phase d'exécution, les utilisateurs peuvent dévier du schéma initial du workflow conçu. Le modèle de précédé issu du processus de découverte de procédé peut être utilisé pour une «Delta analyse» (*c.f.* figure 5.9). Cette «Delta analyse» entre le modèle de procédé initial et découvert, nous permet de surveiller ces déviations. L'analyse de ces déviations est fondamentale pour toute nouvelle phase de re-ingénierie. En effet, une déviation peut devenir une pratique courante plutôt que d'être une exception rare. Dans ce cas, la correction et la fiabilité du modèle de procédé initial sont incertaines et une phase de re-ingénierie, en se basant sur les disparités entre les deux modèles, est exigée.

Figure 5.9 Delta analyse pour la re-ingénierie des procédés



La «Delta analyse» nécessite une technique pour comparer les deux modèles de procédés. Bien que les techniques de comparaison de modèles de procédés ne constituent pas le coeur de notre travail mais plutôt un outil, nous allons, dans la suite, présenter une vue d'ensemble des solutions existantes que nous pouvons utiliser dans la «Delta analyse». En jugeant par le grand nombre de notions d'équivalence [vGW96] cette tâche (c.à.d, la comparaison de modèle de procédés) est loin d'être insignifiante. La plupart des approches de comparaison de modèle de procédés propose «une technique de mappage nœud par nœud» plutôt qu'une comparaison des différences dans le comportement, c.à.d, l'intérêt se porte sur des différences syntaxiques plutôt que des différences sémantiques.

Cependant, d'un point de vue théorique, il y a au moins deux approches qui incorporent également une comparaison comportementale. La première approche consiste à utiliser «l'héritage du comportement» [BvdA01; vdAB01a]. La deuxième approche se base sur le calcul les «régions de changement» des procédés [vdA01; EKR95]. En se basant sur les notions d'«héritage du comportement» définies dans [BvdA01], Van der aalst et autres ont développé la notion du plus grand diviseur commun et de plus petit multiple commun de deux procédés [vdAB01a] comme outil de comparaison. Quant au calcul d'une «région de changement» [vdA01; EKR95], il est déterminée en comparant les deux modèles de procédés et en étendant les régions qui ont été changées directement par les parties du procédés qui sont également affectées par le changement

venant de l'autre procédé, c.à.d, les parties affectées syntaxiquement sont étendues avec les parties sémantiquement affectées pour produire les «régions de changement».

Indépendamment de la technique de comparaison choisie, le résultat de la Delta analyse révèle les disparités entre les deux modèles. Ces disparités peuvent être exploitées pour motiver les utilisateurs à coller au modèle de procédés initialement conçu si les disparités n'expriment pas une évolution réelle du procédé (option 1) ou pour corriger et améliorer le modèle de procédé pour coller au mieux à la «réalité» de la phase d'exécution (option 2). Dans la suite, nous proposons un ensemble de règles de correction et de suggestions d'amélioration qui s'inscrivent dans la deuxième option.

5.5.2 Correction du comportement transactionnel

L'ensemble de règles de correction (*c.f.* règle 5.1), que nous proposons, permet de supprimer les mécanismes de recouvrement initiaux inexistants dans le workflow découvert. Ces mécanismes erronés peuvent être coûteux pour le système qui doit fournir les moyens nécessaires pour les supporter.

RÈGLE 5.1 (SUPPRESSION DE COMPORTEMENTS TRANSACTIONNELS INDÉSIRABLES)
$\forall$ activité transactionnelle $a \in Wf$ un workflow transactionnel,

1. **S1** : Si $\nexists ITR_{a_{decouvert}} \wedge a^{c}_{initiale}$

   Supprimer tout les mécanismes de recouvrement de $a_{initiale}$ (propriété rejouable ou/et dépendances d'alternatives, de suspension ou d'annulation : $a^{\not c}_{corrige}$;

2. **S2** : Soit $b \in WF|TSDF(b_{decouvert}, a_{decouvert}) \geq 0 \wedge a^{\not c}_{decouvert} \wedge depAnl(a_{initiale}, b_{initiale}) = VRAI$

   Supprimer toutes dépendances d'annulation vers des activités non parallèles : $depAnl(a_{corrige}, b_{corrige}) = FAUX$;

3. **S3** : Soit $b \in WF|TSDF(b_{decouvert}, a_{decouvert}) \geq 0 \wedge a^{\not c}_{decouvert} \wedge depSus(a_{initiale}, b_{initiale}) = VRAI$

   Supprimer toutes dépendances de suspension vers des activités non parallèles : $depSus(a_{corrige}, b_{corrige}) = FAUX$;

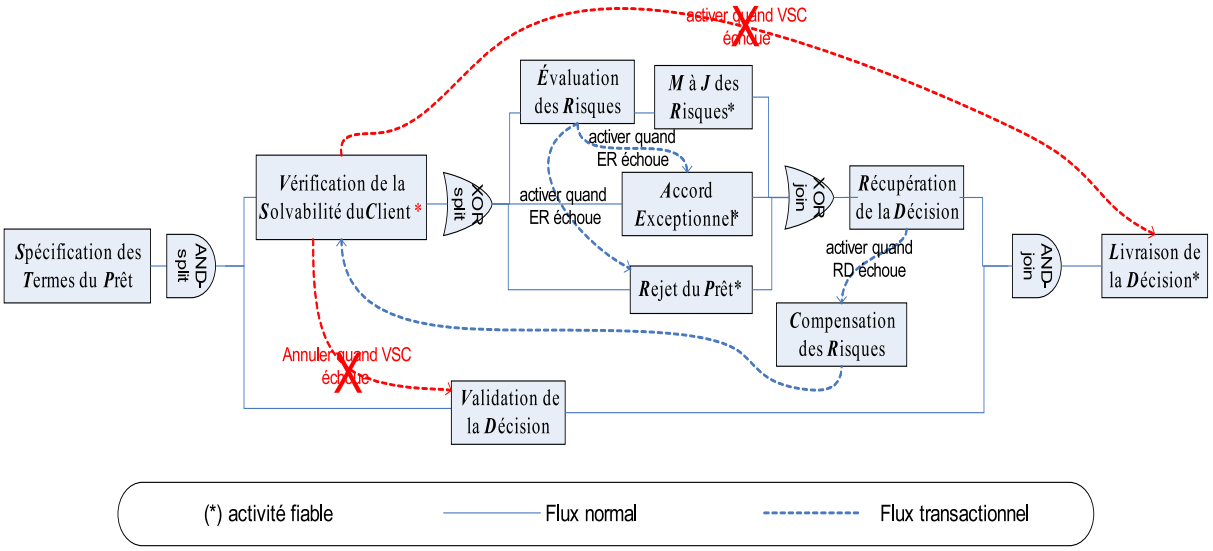
La première règle **S1** exprime le fait que lorsqu'on découvre qu'une activité n'échoue jamais alors tout mécanisme de recouvrement spécifié initialement qu'il soit la propriété transactionnelle rejouable ou les dépendances transactionnelles d'alternative d'annulation ou de suspension, qui en découlent, ne sont plus nécessaires au bon déroulement du workflow et leur maintien peut causer un coût supplémentaire, d'où la proposition de les supprimer. La deuxième et la troisième règle **S2** et **S3** sont inspirées des règles **R1** et **R2** des relations sémantiques entre `flot de contrôle` et `flot transactionnel` (*c.f.* règle 4.1). Elles indiquent que des dépendances de suspension ou d'annulation ne peuvent exister qu'entre des activités parallèles. Ainsi, si on découvre que les activités qui ne sont pas en réalité en concurrence (ce cas peut se produire pour les activités des flux parallèles des patrons `M-out-of-N` et `choix multiples` du fait du comportement de concurrence partielle) alors on propose d'éliminer ce comportement transactionnel qui peut être coûteux à supporter et à gérer.

Exemple : Nous nous proposons d'illustrer l'applicabilité de nos règles de suppression de comportements transactionnels indésirables sur notre exemple de workflow de prêt bancaire. Supposons que nous ayons découvert que VSC n'échoue jamais. Ceci constitue une différence entre le modèle découvert et le modèle initialement conçu qui suppose que VSC échoue. Alors nous pouvons conclure par :

- l'application de **S1**, qu'il n'y a pas besoin que VSC soit recouvrable. Nous supprimons ainsi la dépendance transactionnelle alternative entre VSC et LD.
- l'application de **S2**, qu'il n'y a plus besoin d'annuler VD à l'échec de VSC. Nous enlevons ainsi la dépendance d'annulation entre VSC et VD.

La figure 5.10 illustre ces corrections sur notre exemple de workflow de prêt bancaire en les signalant en rouge.

Figure 5.10 correction du `flot transactionnel` du workflow de prêt bancaire.



5.5.3 Suggestions de mécanismes de recouvrement

Par ailleurs, nous définissons aussi des règles qui proposent des suggestions de mécanismes de recouvrement aux concepteurs pour des activités dont on découvre qu'elles ont échoué mais dont on n'a pas spécifié initialement des mécanismes de recouvrement (*c.f.* règle 5.2). Ces règles s'inspirent des relations sémantiques entre `flot de contrôle` et `flot transactionnel` définies dans la règle 4.1. Notons que toute activité, qui échoue, ne nécessite pas nécessairement un mécanisme de recouvrement. Le choix de lui développer un mécanisme de recouvrement dépend des choix métiers et conceptuels du concepteur mais doit aussi respecter la relation **R5** définie dans la règle 4.1. Cette relation définit pour un workflow, indépendamment des choix métiers et conceptuels du concepteur, un ensemble d'activités qui doivent être atomiques ou recouvrables. Ainsi, si on découvre qu'une activité initialement conçue comme atomique peut échouer, on doit lui spécifier un mécanisme de recouvrement pour qu'elle soit recouvrable.

Les règles **A1.1**, **A1.2** et **A1.3** suggèrent trois propositions différentes pour répondre au besoin de mécanismes de recouvrement d'une activité qui n'était pas recouvrable dans la phase

RÈGLE 5.2 (SUGGESTION DE MÉCANISMES DE RECOUVREMENT ADDITIONNELS)
Pour chaque activité transactionnelle $a \in Wf$ un workflow transactionnel, qui est initialement non recouvrable $a^{\not c}_{initial}$ mais qu'on découvre qu'elle échoue, nous proposons les suggestions suivantes pour qu'elle soit recouvrable $a^{c}_{corrige}$:

1. **A1.1** : Si l'activité a est neutre ou idempotente

   Suggérer que a soit rejouable : $a^{r}_{corrige}$ OU exclusive ;

2. **A1.2** : Soit $b \in WF$ une activité représentant un point cohérent pour a $|((a_{decouvert} \lhd b_{decouvert}) \vee TSDF(b_{decouvert}, a_{decouvert}) \neq -1$

   Suggérer un mécanisme de recouvrement en avant :
   $depAlt(a_{corrige}, b_{corrige}) = VRAI$ OU exclusive ;

3. **A1.3** : Soit $b \in WF$ une activité représentant un point cohérent pour a $|(b_{decouvert} \lhd a_{decouvert} \wedge b^{\not c}_{decouvert} \wedge (\nexists c \in WF \; | c_{decouvert} \lhd b_{decouvert} \lhd a_{decouvert})$

   Suggérer un mécanisme de recouvrement en arrière :
   $depAlt(a_{corrige}, b_{corrige}) = VRAI$;

4. **A2** : $\forall b \in WF \; | TSDF(b_{decouvert}, a_{decouvert}) = -1 \wedge b^{\not c}_{decouvert}$

   Suggérer une dépendance d'annulation vers les activités parallèles :
   $depAnl(a_{corrige}, b_{corrige}) = VRAI$;

5. **A3** : $\forall b \in WF \; | TSDF(b_{decouvert}, a_{decouvert}) = -1$

   Suggérer une dépendance de suspension vers les activités parallèles :
   $depSus(a_{corrige}, b_{corrige}) = VRAI$;

de conception initiale. La règle **A1.1** propose que l'activité soit rejouable au cas où elle est neutre. Le fait qu'elle soit neutre (c.à.d. que son échec ne cause pas un dérèglement de l'exécution des autres activités) ou pas revient au jugement personnel du concepteur qui dépend de ces choix métiers et conceptuels. La règle **A1.2** suggère une dépendance alternative en avant dans le cas de l'existence d'une activité qui représente un point cohérent en avant pour l'activité échouée. Cette règle **A1.1** respecte la relation sémantique **R6** en indiquant le fait que l'activité représentant ce point cohérent ne soit pas en concurrence avec l'activité échouée. La règle **A1.3** suggère une dépendance alternative en arrière dans le cas de l'existence d'une activité qui représente un point cohérent en arrière pour l'activité échouée. Cette règle **A1.3** respecte de la relation sémantique **R7** en s'assurant qu'il n'existe pas d'activités pivots qui se trouvent entre le point cohérent et l'activité échouée. De même, notons ici que l'attribution des points cohérent et leurs localisations dépendent des choix métiers et conceptuels du concepteur.

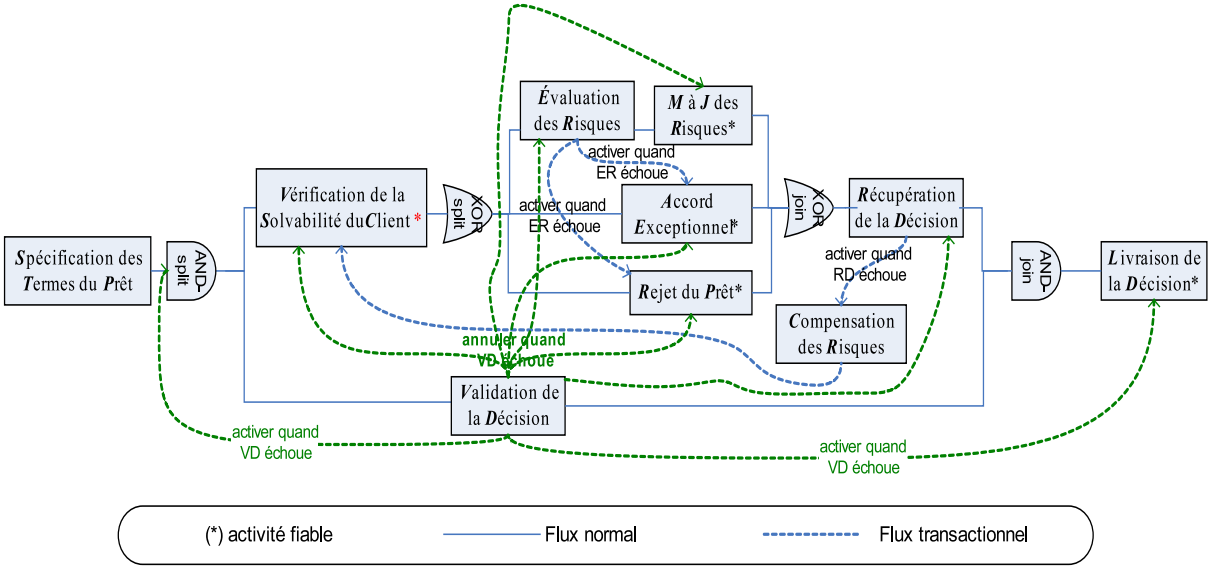
Quant aux deuxièmes et troisièmes règles, elles suggèrent des dépendances transactionnelles de suspension et d'annulation. Ces dépendances transactionnelles peuvent être nécessaires pour «accompagner» des mécanismes de recouvrement initialement conçus ou ajoutés durant cette phase de re-ingénierie. La règle **A2** déduit des relations sémantiques **R1** et **R3** le fait qu'on peut proposer une dépendance d'annulation de l'activité découverte échouée vers les activités qui s'exécutent en parallèle avec elle et qui sont non atomiques. Finalement, la règle **A3** déduit de la relation sémantique **R2** le fait que l'on peut proposer une dépendance de suspension de l'activité découverte échouée vers les activités concurrentes.

Exemple : Nous nous proposons d'illustrer l'applicabilité de nos règles de suggestions des mécanismes de recouvrement additionnels (*c.f.* règle 5.2) sur notre exemple de workflow de prêt bancaire. Supposons que nous ayons découvert que VD peut échouer. Ceci constitue une différence entre le modèle découvert et le modèle initialement conçu qui suppose que VD n'échoue jamais. Si on suppose que VD n'est pas neutre, nous pouvons recommander au concepteur les solutions suivantes :

- Par l'application la relation **R5** définie dans la règle 4.1, nous devons proposer un mécanisme de recouvrement pour VD. Ainsi nous pouvons suggérer :
  - Par l'application de **A1.2**, nous pouvons suggérer un recouvrement en avant où le point cohérent devrait être localisé juste avant LD. OU exclusive
  - Par l'application de **A1.3**, nous pouvons suggérer un recouvrement en arrière pour VD où le point cohérent se trouverait juste après STP, si les activité qui sont concurrentes à VD (VSC, ER, MJR, AE, RP et RD) ne sont pas des activités pivots.
- Par l'application de **A2**, nous pouvons suggérer de spécifier des dépendances d'annulation entre VD d'une part et (VSC, ER, MJR, AE, RP et RD) d'autre part.

La figure 5.11 illustre ces suggestions sur notre exemple de workflow de prêt bancaire en les marquant en vert. Ces suggestions peuvent être appliquées en entier et en partie selon les choix conceptuels des concepteurs.

Figure 5.11 Correction du `flot transactionnel` du workflow de prêt bancaire.



5.5.4 Synthèse

Dans cette section, nous nous sommes intéressés à la découverte et l'amélioration du comportement transactionnel du workflow. En utilisant les résultats découverts et en les comparant au schéma de workflow initialement conçu, nous proposons un ensemble de règles pour l'amélioration et la correction du comportement transactionnel du workflow. Les disparités entre les deux modèles conçus et découverts peuvent exprimer à la fois des erreurs de choix conceptuels du modèle conçu comme des nouveaux besoins d'évolution. Ainsi, Cette étape nous permet de prendre en

compte les nouvelles contraintes qu'on a pues voir dans l'exécution dans une nouvelle phase de re-conception pour l'amélioration et la correction des mécanismes de recouvrement. Nous procédons par un ensemble de suggestions respectant certaines normes de «bonnes conduites conceptuelles» exprimées dans les relations sémantiques décrites dans notre modèle de workflow transactionnel. De plus, pour tenir compte des besoins spécifiques des concepteurs, nous procédons d'une façon interactive. En effet, les corrections induites par ces suggestions ne sont pas faites d'une façon automatique mais plutôt interactive avec les concepteurs pour qu'ils puissent préciser leurs besoins spécifiques.

En conclusion, cette section nous a permis de traiter de la cinquième interrogation de notre problématique : «Comment utiliser les résultats du processus de découverte pour améliorer le traitement des échecs d'exécution et optimiser la fiabilité et le coût des mécanismes de recouvrement dans le cadre du processus de re-ingénierie des procédés ? ?». Cet objectif a été atteint en proposant d'intégrer la phase de découverte dans le processus de re-ingénierie du procédé, en comparant le modèle découvert au modèle initialement conçu, en exploitant les disparités entre ces deux modèles dans la re-conception, et en spécifiant un ensemble de règles qui permettent de générer un ensemble de suggestions dans le but de la correction et de l'amélioration du comportement transactionnel du workflow.

5.6 Conclusion

Dans ce chapitre, nous avons présenté une approche originale de re-ingénierie des procédés, en particulier le comportement transactionnel, en nous basant sur des techniques de découverte de procédé. Cette approche permet d'améliorer les mécanismes de recouvrement d'un workflow par l'analyse des traces d'exécutions. L'avantage de cette approche est qu'elle repose sur des faits réels (par observation des exécutions) et non sur des hypothèses pouvant ne pas coïncider avec la réalité. Cette particularité nous permet de corriger les erreurs potentielles des hypothèses de la phase de conception initiale et surtout de coller au mieux à l'évolution du schéma conceptuel procédé.

Notre approche a été formellement démontrée tout au long des différents étapes qui la composent. Nous avons spécifié la structure des traces d'exécutions de workflow qui sont l'unique entrée pour nos techniques de découverte de workflow. Nous avons démontré les conditions minimales et suffisantes que ces traces d'exécutions doivent vérifier pour nos techniques de découverte.

Notre approche de découverte de workflow se base sur des techniques d'analyse statistique des traces d'exécutions. Nous avons commencé par la découverte du `flot de contrôle` en décrivant un algorithme dynamique qui bâtit le résultat final (le workflow final) à partir de résultats locaux (les patrons de workflow). Par la suite, nous avons découvert le `flot transactionnel` du workflow en capturant les dépendances transactionnelles intra-activité et inter-activités décrivant respectivement les propriétés transactionnelles des activités et les dépendances transactionnelles entre les activités.

Dans la phase de re-ingénierie, nous avons généré un ensemble de suggestions permettant de corriger et d'améliorer le comportement transactionnel du workflow en se basant sur les disparités entre le modèle découvert et le modèle initialement conçu. Ces suggestions sont décrites par des règles qui respectent les relations sémantiques entre flux de contrôle et flux transactionnel de notre modèle de workflow transactionnel et permettent d'améliorer le traitement des erreurs d'exécution et d'optimiser la fiabilité et le coût des mécanismes de recouvrement. Par ailleurs, notre approche permet de tenir compte des besoins des utilisateurs en les impliquant dans notre démarche de re-ingénierie de procédés, d'abord, en reprenant les besoins qu'ils ont pus exprimer

lors de la phase d'exécution ,ensuite, en procédant d'une façon interactive avec eux à travers l'ensemble de suggestions générées par les règles décrites ci-dessus lors de la phase de correction et d'amélioration, pour mieux intégrer leurs besoins spécifiques.

Bien que nombreux travaux aient été entrepris dans le découverte du `flot de contrôle` des workflows, peu de propositions ont été décrites dans la littérature sur l'issue de l'exploitation des résultats des algorithmes de découverte de workflow pour la re-ingénierie des procédés. On peut citer par exemple, les travaux récents de van der Aalst et autres [vdAdM05] pour des vérifications rétrospectives concernant des violations de la sécurité qui montrent comment un algorithme de découverte de procédé spécifique peut être employé pour supporter des efforts de détection d'intrusion ou d'empêchement de fraudes. Dans le même contexte, Rozinat et autres [RvdA05] s'intéresse à l'utilisation potentielle de la découverte de procédé pour mesurer l'*alignement des procédés*, c.à.d, comparer le vrai comportement du système d'information ou de ses utilisateurs au comportement prévu ou attendu. Cependant, ce travail ne décrit pas comment utiliser les résultats de cet *alignement des procédés* pour améliorer ou corriger le procédé. Par ailleurs, il existe d'autres travaux relatifs au traitement du comportement des workflows à l'exécution [SCSD02; GCC+04] en exploitant les traces d'exécutions. Ils décrivent essentiellement des outils qui fournissent des mécanismes de calcul de performance et de contrôle dynamique de l'exécution.

Il y a eu beaucoup de travaux formelles pour l'analyse à priori du modèle de workflow, utilisant par exemple des réseaux de Pétri ou les algèbres de procédé [FG94; FG94; CWBH+03]. Au contraire de notre approche ils se concentrent typiquement sur la vérification d'une conception plutôt que d'analyser le comportement réel. À notre connaissance, nous sommes des précurseurs dans l'exploitation des résultats découverts à partir des traces d'exécutions pour l'amélioration du comportement transactionnel. En plus, nous comptons parmi les initiateurs de la découverte du comportement transactionnel. C'est pourquoi la discussion, que nous avons menée dans la section 5.3.6, a porté **uniquement** sur la comparaison de nos techniques de découverte du `flot de contrôle` par rapport aux approches existantes, par manque ou par absence d'autres techniques ou approches dans la littérature portant sur la découverte et l'amélioration du comportement transactionnel.

Chapitre 6

Mise en œuvre

Table des matières

6.1 Introduction

Dans ce chapitre, nous développons les travaux de mise en œuvre que nous avons effectués pour la validation de notre proposition de fiabilisation des exécutions et de découverte de workflow par l'analyse des traces d'exécutions. En premier, nous présentons, dans la section 6.2, le système de gestion de workflows `Bonita` que nous avons étendu pour y intégrer le comportement transactionnel des activités. Par la suite, dans la section 6.3, nous présentons les outils de collecte de traces d'exécutions que nous avons implantés. Pour ce faire, nous avons proposé deux solutions : la première consiste à greffer une API dans `Bonita` pour la collecte de traces d'exécution réelles, la deuxième consiste à générer des traces d'exécutions simulées. Finalement, nous avons implanté, dans la section 6.4, nos algorithmes de découverte de workflow ; D'abord dans un prototype que nous avons appelé `Workflowminer` [GBG06], ensuite dans le plug-in «`Workflow patterns miner`» [GG06] intégré dans la plateforme `ProM`.

Dans un souci de simplicité et de clarté, nous avons choisi de séparer les détails de construction logicielle et de tests de validation du chapitre courant, et de les mettre dans deux annexes

différentes. D'une part, l'architecture logicielle des différentes applications développées fait l'objet d'une présentation plus détaillée dans l'annexe C à travers leurs diagrammes UML. D'autre part, l'annexe D contient un ensemble exhaustif d'exemples de workflows que nous avons utilisés pour appliquer et tester les outils de collecte de traces d'exécutions et de découverte de workflow que nous proposons.

6.2 Environnement de conception de workflow transactionnel

L'intégration du comportement transactionnel dans le système de gestion de workflow `Bonita` a été sujet d'un travail de développement mené récemment dans notre équipe de recherche dans le cadre d'une application de composition de services Web transactionnels [BPG06]. Dans la suite, nous nous basons sur ce travail que nous avons adapté au contexte workflow. Nous proposons, au début une description brève de l'architecture générale de `Bonita` (*c.f.* section 6.2.1). Ensuite, nous illustrons comment le système de gestion de workflows `Bonita` a été étendu, via l'utilisation de plug-ins «transactionnels», pour qu'il puisse supporter le comportement transactionnel (*c.f.* section 6.2.2). Finalement, nous présentons dans la section 6.2.3 quelques exemple de ces plug-ins «transactionnels».

6.2.1 Présentation de `Bonita`

6.2.1.1 Vue générale de `Bonita`

`Bonita` [MC03] est un système de workflow coopératif de troisième génération conforme aux recommandations de WfMC, créé par le Consortium ObjectWeb et Ecoo, Open Source et téléchargeable sous la license de LGPL[26]. Il permet de spécifier, d'exécuter et de contrôler des procédés coopératifs. Il fournit un ensemble complet d'outils graphiques intégrés et basés sur JFC/Swing [Mic], pour exécuter le procédé défini, instantiatier et commander ce procédé, et assurer une interaction entre les utilisateurs et d'autres applications. `Bonita` inclut aussi un navigateur permettant de gérer et de contrôler l'exécution des procédés d'une manière interactive par l'utilisation de l'application «Java Web start» et des technologies des services Web permettent aux concepteurs et aux organisations de le manipuler via le Web.

`Bonita` est implémenté en utilisant J2EE Enterprise Java Beans [Sun02] qui fournit un environnement flexible et portable. La figure 6.1 illustre l'architecture générale de `Bonita` :

La couche de persistance de `Bonita` : Les principaux éléments de la couche persistance de `Bonita` sont les *projets*. Un *projet* représente un procédé de workflow. Chaque procédé de workflow va contenir essentiellement des informations relatives au nom du procédé, à l'utilisateur créateur, à l'état de l'exécution du procédé, etc. et un grand nombre de données associées : les activités, les connecteurs, les données du procédé, les participants, etc.

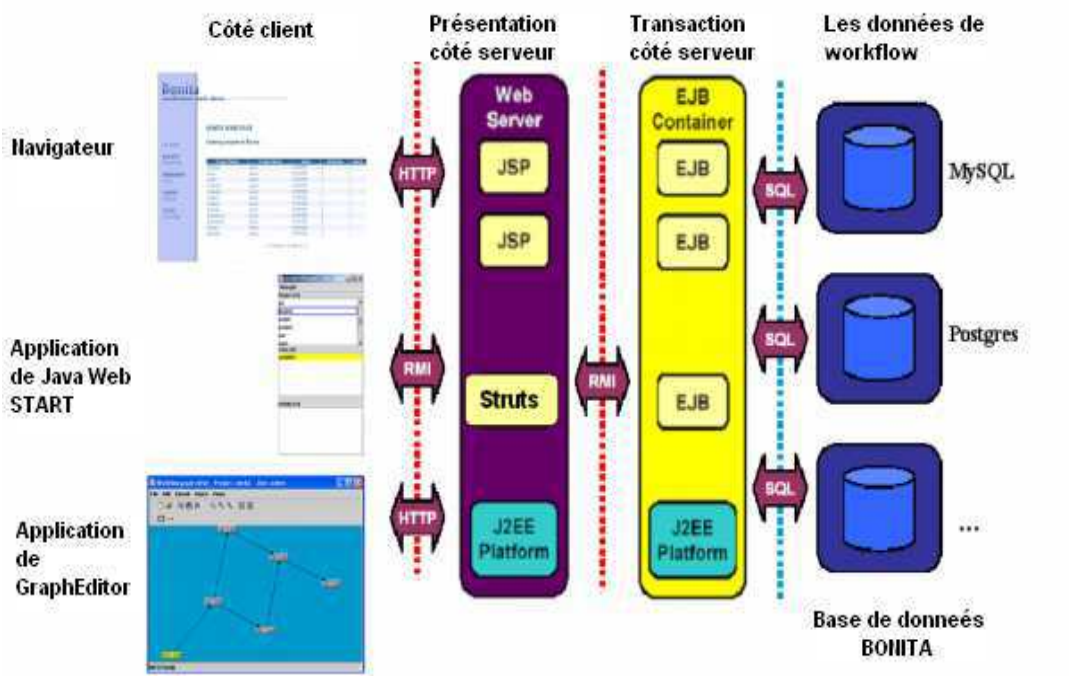
Afin de séparer les données du *projet* des éléments qui composent le *projet* lui même, un bean[27] portable a été développé pour assumer les relations avec les autres entités qui composent la représentation d'un workflow dans `Bonita` : activités, connecteurs, utilisateurs, rôles, propriétés, etc.

L'éditeur graphique : Dans `Bonita`, le composant GraphEditor permet au concepteur de représenter et de visualiser le procédé de workflow en spécifiant les activités et les connecteurs. L'utilisateur peut spécifier les attributs d'une activité (type, description, mode d'exécution, date limite, rôle, etc.). Ce composant permet de créer des projets et de gérer les différentes activités

[26] Lesser General Public License

[27] Les beans (ou Javabeans) sont des composants logiciels écrits en Java

Figure 6.1 Architecture générale du système de workflow `Bonita`.



du projet. Une fois le projet créé et le schéma du workflow défini, les utilisateurs peuvent se connecter afin de définir son état d'avancement. En effet, lorsqu'un utilisateur se connecte, il voit apparaître trois sous fenêtres. La première contient la liste des projets en cours. La deuxième contient la liste des tâches à effectuer. Et la troisième partie contient la liste des activités en cours d'exécution. L'utilisateur peut alors, en utilisant le menu contextuel, modifier l'état de l'activité ou du projet sélectionné.

Le moteur d'exécution de `Bonita` : Le moteur d'exécution de `Bonita` est constitué du Bean «EngineSession», des modes d'exécution et de la liste des tâches. Le bean «EngineSession» définit toutes les opérations (actions) d'un procédé ou d'une activité. Le moteur d'exécution est responsable de gérer les différents modes d'exécution. Il offre aussi une gestion flexible des données et une exécution flexible du workflow. La gestion flexible des données permet aux activités d'échanger des résultats intermédiaires permettant l'anticipation d'exécution des activités. Par ailleurs, le moteur d'exécution de `Bonita` introduit aussi le concept de «hook» d'activité. Les «hooks» dans `Bonita` sont des unités de code source qui peuvent être exécutées par le moteur pendant l'exécution. Elles sont associées aux activités des procédés de workflow. Les «hooks» doivent être écrites en Java ou dans l'un des langages scripts objet disponibles dans `Bonita`. L'utilisateur peut associer des «hooks» à différents moments du cycle de vie de l'exécution d'une activité.

L'interface Web : L'interface Web est très pratique car accessible depuis n'importe quel poste possédant un navigateur Web. Elle permet de visualiser l'ensemble des projets (procédés) et des activités relatives à l'utilisateur connecté. L'utilisateur peut ensuite, en clinquant sur un projet ou une activité, accéder à une page contenant les informations relatives. A partir de cette page, l'utilisateur peut modifier certaines caractéristiques comme le rôle, la date de fin, la description et la capacité de l'activité à être anticipée.

Les services communs de `Bonita` : `Bonita` intègre un grand nombre de services de contrôle :

- Le service de messagerie JMS de `Bonita` notifie la définition et l'exécution des changements dans un procédé de workflow. Chaque interaction d'un utilisateur est notifiée au noyau de `Bonita` et lance un événement JMS.
- `Bonita` utilise JMX[28] pour imposer des dates limites aux activités. Ce qui permet de notifier à l'utilisateur qu'une activité ne se termine pas à sa date attendue.
- Le service de notification «Jabber» de `Bonita` permet aux utilisateurs de recevoir des notifications en temps réel et échanger différents types de messages.

6.2.2 Extension de `Bonita` via des plug-ins

6.2.2.1 Approche adoptée

Initialement, `Bonita` a été conçu pour supporter des exécutions flexibles, en particulier l'échange des résultats intermédiaires et l'anticipation. Afin d'intégrer le comportement transactionnel, on a étendu `Bonita` pour qu'il puisse prendre en compte les propriétés et les dépendances transactionnelles des activités.

Une première approche, consistait à utiliser les «hooks», puisqu'ils permettent d'exécuter un code pendant l'exécution, en généralisant leur utilisation pour spécifier les mécanismes de recouvrement en cas d'échec d'exécution ou définir les propriétés transactionnelles des activités. Cependant, cette approche ne prend pas en considération les différents modèles de procédés qui peuvent être envisagés dans un contexte plus général. Chaque modèle utilise un ensemble particulier d'états. La généralisation des «hooks», permettra la personnalisation du comportement d'une activité, mais pas l'implémentation d'un modèle de procédé particulier.

Prenant en compte cette considération, nous avons adopté une autre approche. Cette approche consiste à identifier les aspects transactionnel du modèle de procédé qu'on peut personnaliser et de les séparer dans des plug-ins «transactionnels». Chaque modèle de workflow transactionnel doit définir les aspects spécifiques lui correspondant et les implémenter dans des plug-ins «transactionnels». Le moteur doit être changé pour qu'il soit le plus indépendant possible des modèles de procédé. Il doit demander aux plug-ins «transactionnels» les informations dépendantes du comportement transactionnel.

Dans ce qui suit, nous décrivons la nouvelle architecture en détaillant les plug-ins et les modifications apportées au moteur de `Bonita`. Nous étudions, en particulier, les différents composants de `Bonita` afin de déterminer les éléments et l'architecture d'un plug-in.

6.2.2.2 Éléments relatifs au plug-in

Le plug-in détermine l'état suivant d'un nœud[29] et doit être capable d'ajouter de nouveaux états à ceux qui sont définis par défaut. Nous présentons, en particulier, les états et les opérations des nœuds et nous précisons les éléments de base (qui vont exister indépendamment du plug-in utilisé) et les éléments nécessaires pour construire un plug-in.

États des nœuds : les états des nœuds de base sont :

- *initial* : Initialement un nœud est dans l'état *initial*. Dans cet état, sa condition d'activation n'est pas vérifiée. Tant que cette condition est évaluée à faux, le nœud reste dans cet état *initial*.
- *ready* : Un nœud atteint cet état quand sa condition d'activation devient vraie.

[28] Java Management Extensions

[29] un nœud représente une étape dans un procédé qui peut être une activité dans un contexte de workflow ou un service dans un contexte de services Web.

- *executing* : Un nœud atteint cet état quand il sera choisi pour être exécuté après avoir été activé.
- *suspended* : Un nœud atteint cet état quand il sera suspendu en cours d'exécution. Son exécution peut être reprise par la suite.
- *terminated* : Un nœud atteint cet état quand il se termine avec succès.
- *failed* : Un nœud atteint cet état quand son exécution échoue.
- *expired* : Un nœud atteint cet état quand sa date limite s'est expirée.
- *dead* : Un nœud atteint cet état quand il sera abandonné.
- *Anticipating* : Un nœud, automatique, atteint cet état quand il peut anticiper son exécution alors que ses activités prédécesseurs sont encore en cours d'exécution (se trouvent dans l'état *«executing»*).
- *Anticipatable* : Un nœud, qui n'est pas automatique, atteint cet état si il peut anticiper son exécution et sa condition d'anticipation devient vraie. Ce nœud reste dans cet état jusqu'à un participant le choisit et le fait ainsi transiter vers l'état *«anticipating»*.
- *Anticipation Suspended* : Un nœud atteint ce état quand il est suspendu alors qu'il est dans l'état *«anticipating»*. Son exécution anticipée peut être reprise en appelant l'opération *«resume»*.

Opérations des nœuds : les opérations des nœuds de base sont :

- *start()* : Cette opération permet d'initialiser un nœud. L'état d'un nœud après un appel à cette opération est *«initial»*.
- *activate()* : cette opération permet d'activer un nœud quand sa condition d'activation devient vraie. Son appel le fait transiter vers l'état *«ready»*.
- *suspend()* : Cette opération permet de suspendre l'exécution d'un nœud. Son appel le fait transiter vers l'état *«suspended»*.
- *resume()* : cette opération permet de reprendre l'exécution d'un nœud suspendu.
- *terminate()* : cette opération permet de terminer l'exécution d'un nœud avec succès. Son appel le fait transiter vers l'état *«terminated»*.
- *deactivate()* : cette opération est appelée quand la condition d'activation d'un nœud est évaluée à faux.
- *cancel()* : cette opération permet d'annuler un nœud quand sa condition d'activation ne sera jamais vérifiée. Son appel le fait transiter vers l'état *«dead»*.
- *Change to Failed* : c'est une opération spéciale qui est appelée par le moteur quand l'exécution d'un nœud échoue. Par défaut, l'appel à cette opération fait passer le nœud correspondant à l'état *«failed»*.
- *«Activate Anticipation»* : cette opération est appelée sur un nœud quand sa condition d'anticipation devient vraie. Son appel le fait transiter vers l'état *«anticipating»* si le nœud est automatique et vers l'état *«anticipatable»* sinon.

Comportements des nœuds : le comportement d'un nœud est décrit par un tableau à transitions d'états. Il n'y a pas de comportement de base (c.à.d tous les comportements doivent être définis dans le plug-in). Selon la version actuelle de `Bonita`, il y a quatre comportements possibles qui sont le résultat de la combinaison de l'anticipation et de l'exécution automatique d'activité.

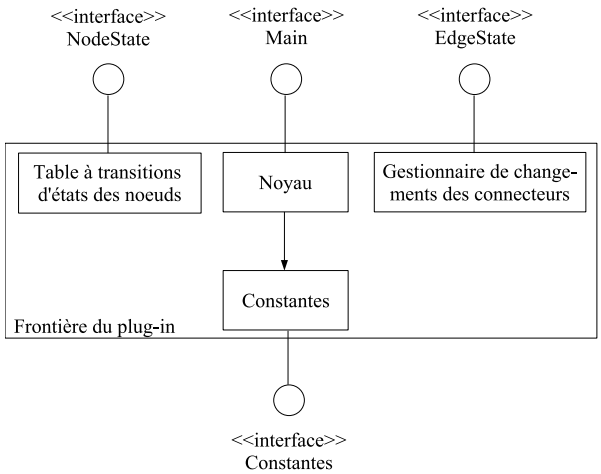
Propriétés personnalisées : en plus des propriétés générales des workflows, le plug-in doit être capable de gérer des propriétés personnalisées pour tout un workflow ou pour une activité particulière. Les propriétés personnalisées peuvent être utilisées pour sauvegarder des données correspondantes au comportement spécifique du workflow ajouté par le plug-in.

6.2.2.3 Architecture d'un plug-in

L'architecture du plug-in de base est illustrée par la figure 6.2 et elle est détaillée à la figure C.5 de l'annexe C. Il est important de noter que la structure interne d'un plug-in n'est pas restreinte à cette architecture. La seule contrainte à respecter est de préserver les interfaces externes et leurs sémantiques que nous expliquons ci-dessous. Dans ce plug-in, nous pouvons distinguer 4 modules principaux :

- **Le noyau :** Ce module permet au moteur d'accéder à tous les éléments d'un plug-in. Afin d'intégrer un modèle particulier, le moteur doit obtenir une instance du noyau du plug-in correspondant dont l'interface est connue. Cette instance permettra au moteur d'exécution de `Bonita` d'accéder aux éléments spécifiques à ce modèle. Les éléments retirés par le noyau peuvent être gérés par le moteur parce qu'ils implémentent des interfaces connues.
- **Les tableaux à transitions d'états des nœuds :** Les types de comportement des nœuds sont représentés par des tableaux à transitions d'états. L'interface *«NodeState»* définit une opération qui retourne l'état du suivant du nœud après l'application d'une opération donnée sur ce nœud. L'implémentation correcte de l'interface *«NodeState»* pour un type de nœud particulier est obtenue via l'élément noyau en exécutant des méthodes dont les signatures sont pré-définies et connues.
- **Le gestionnaire des changements d'états des connecteurs :** Ce module aide le moteur d'exécution à déterminer l'opération à appliquer à un nœud quand ses connecteurs entrants changent d'états. Le changement d'état des connecteurs entrants a un impact sur la condition d'activation, qui peut nécessiter l'application d'une opération sur le nœud. Comme le module des tableaux à transitions d'états, l'interface *«EdgeState»* fournit une méthode qui indique l'opération nécessaire auprès d'un nœud quand ses connecteurs changent d'états.
- **Les constantes :** Ce module définit les états, les opérations, les types et les comportements des nœuds. Les deux modules précédents de l'architecture s'intéressent au comportement du modèle de procédé. Ce module s'intéresse aux définitions statiques du modèle de procédé.

Figure 6.2 Architecture de base du plug-in.



L'interface d'un plug-in est un groupe de classes abstraites qui définit tous les états, les opérations et les types de base. Elle définit aussi les méthodes pour faciliter l'accès à ces concepts par le moteur d'exécution. Tout plug-in doit étendre ces classes abstraites en ajoutant les concepts

spécifiques lui correspondant et en réécrivant les méthodes de base.

6.2.2.4 Adaptation du moteur d'exécution

Certains éléments du moteur doivent être modifiés afin de supporter les plug-ins.

Les «hooks» : Les états peuvent être redéfinis par un plug-in. Ainsi, puisque les «hooks» sont étroitement liés aux changements d'états, leur gestion doit être redéfinie. Il est possible de définir la relation entre les états et les «hooks». Pour chaque état d'un nœud, il y aura deux «hooks» qui seront exécutés respectivement avant et après que l'état du nœud change.

Par exemple, la gestion des échecs peut être faite à l'aide d'une «hook» qui s'exécute avant que le nœud passe à l'état échoué (nous appelons cette «hook» *«beforeChangingToFAILED»*). Cette hook peut exécuter une alternative de l'activité du nœud ou elle peut la rejouer (selon les propriétés de ce nœud).

Afin de contrôler l'exécution d'une «hook», des pre et des post conditions sont définies. Elles visent à contrôler l'état avant et après l'exécution d'une «hook» afin d'interdire certains changements (e.g les changement d'état d'un nœud durant les «hooks» *beforeChangingTO* qui peuvent interdire l'exécution d'une «hook» *afterChangingTo*). Si une condition d'une «hook» n'est pas vérifiée ou une exception se déclenche durant son exécution, l'état du nœud passe à l'état *«Failed»*.

Gestion de plusieurs plug-ins dans la même installation de `Bonita`. `Bonita` peut gérer plusieurs plusieurs plug-ins. Si un projet utilise des concepts offerts par plusieurs plug-ins, un nouveau plug-in est défini. Ce nouveau plug-in agira comme une façade pour les autres plug-ins qui doivent être implémentés. Un plug-in peut être défini en étendant les classes d'un autre plug-in. Un plug-in de base est défini par défaut. Il devra être étendu par un autre plug-in si nécessaire.

6.2.3 Exemples de plug-ins «transactionnels»

Nous avons développé deux plug-ins «transactionnels» (qui étendent le plug-in de base) pour que `Bonita` puisse supporter notre approche transactionnelle. Ces deux plug-ins sont : le plug-in rejouable et le plug-in alternative.

6.2.3.1 Plug-in «rejouable»

Ce plug-in permet de gérer la propriété *«rejouable»*. Cette propriété est définie en spécifiant le nombre maximal et la date limite de réitération (c.à.d après cette date limite, il n'est plus possible de rejouer l'activité correspondante). Ces informations sont sauvegardées comme deux propriétés d'un nœud.

Par rapport au plug-in de base, le plug-in «rejouable» n'ajoute pas de nouveaux états ni de nouvelles opérations. Cependant, il modifie le tableau à transitions d'états d'une activité *rejouable*. Dans le plug-in de base, l'opération *«changeToFAILED»* fait transiter une activité vers l'état *«Failed»*. Cependant, dans le plug-in «rejouable» cette opération peut conduire à un échec ou à une ré-activation de l'activité (c.à.d passer à l'état *«Ready»*) selon ses propriétés de réitération.

Nous définissons le plug-in «rejouable» en étendant le plug-in de base afin de modifier le tableau à transitions d'états et de garder toutes les autres fonctionnalités. Ainsi, la méthode *«computeState»* de l'interface *«NodeState»* (*c.f.* figure C.6) est réécrite pour qu'elle prenne en considération les deux propriétés de réitération d'une activité lors du calcul de son nouveau état suite à l'application de l'opération *«changeToFAILED»*.

6.2.3.2 Plug-in alternative

Ce plug-in permet de gérer le mécanisme de recouvrement par alternatives pour une activité. Chaque activité peut être définie comme recouvrable par une ou plusieurs alternatives de recouvrement en arrière ou en avant. Une activité recouvrable par alternatives possède un ensemble de points cohérents définissant des chemins d'alternatives, dont la cardinalité varie de 0 à N (zero signifie que l'activité n'est pas recouvrable). Une activité qui a plusieurs chemins d'alternative est dite sélective. Par exemple, une activité A peut être définie comme recouvrable et son ensemble de points cohérents est $Comp_A = \{B, C, D\}$.

Par rapport au plug-in de base, le plug-in «alternative» ajoute un nouvel état *«recovering»* et *«recovered»*. Une activité atteint l'état *«recovering»* quand un de ses chemins d'alternative est en cours d'exécution. Deux opérations peuvent être appelées quand une activité est dans l'état *«recovering»*. La première est l'opération *«terminate»*. L'appel de cette opération désigne la terminaison du recouvrement avec succès. L'activité passe ainsi à l'état *«recovered»* qui est equivalent à l'état *«terminated»* de notre modèle de workflow transactionnel. La deuxième opération est *«cancel»*. L'appel de cette opération désigne l'échec du recouvrement. L'activité passe dans ce cas à l'état *«Failed»*.

Pour les activités sélectives (c.à.d avec plusieurs alternatives), c'est l'utilisateur qui doit choisir l'alternative de recouvrement à exécuter dans le cas où un mécanisme de recouvrement est nécessaire.

6.2.4 Synthèse

Dans cette section, nous avons présenté une implémentation possible du comportement transactionnel décrit de notre modèle de workflow transactionnel. Concrètement, nous avons montré comment nous avons adapté le système de workflow `Bonita`, via différents plug-ins «transactionnels», pour qu'il puisse supporter différents comportements transactionnels de notre modèle de workflow transactionnel. Nous avons détaillé en particulier les plug-ins rejouable et alternative que nous avons définis pour que `Bonita` puisse supporter notre approche transactionnelle. Par ailleurs, l'approche conceptuelle que nous avons choisi pour étendre `Bonita` permet d'intégrer n'importe quel modèle transactionnel.

Notons que, d'une part, l'utilisation des «hooks», deja existants dans l'environnement de conception initial de `Bonita`, pour décrire le comportement transactionnel peut être assimilé à un simple raffinement du langage de workflow initial de `Bonita` (c.à.d. le langage du `flot de contrôle`), donnant lieu ainsi à un langage de conception de workflows transactionnels de type **WF/TR**. D'autre part, notons que, d'un point architectural, la spécification du comportement transactionnel à travers des plug-ins «transactionnels» indépendants de l'architecture conceptuelle originel de `Bonita`, donne lieu à une architecture de type **MTR+MWF** où le modèle de workflow classique MTR est pris en charge par `Bonita` «classique» (c.à.d. avant l'extension), et le modèle transactionnel MWF est pris en charge par les plug-ins «transactionnels». Cette démarche nous permet de respecter les caractéristiques de notre modèle de workflow transactionnel qui sont une classe conceptuelle **WF/TR** et une classe architecturale **MTR+MWF**.

Par ailleurs, notre implantation a été réalisée dans un esprit de changer `Bonita` le moins possible afin de tirer profit de la stabilité de la version courante. En effet, puisque `Bonita` évolue constamment, ayant le moins de modifications possibles dans le moteur facilitera l'adaptation de version la plus récente afin de supporter la nouvelle approche des plug-ins «transactionnels».

6.3 Collecte de traces d'exécutions

Dans cette section, nous présentons l'approche que nous avons adoptée pour récupérer les traces d'exécutions nécessaires à la validation de nos algorithmes de découverte de workflows. Nous avons implanté deux moyens complémentaires pour répondre à cette issue : la collecte de traces d'exécutions réelles et la génération de traces d'exécutions simulées. Le premier objectif a été atteint par l'implantation d'une API greffée à `Bonita` pour collecter les traces d'exécutions des instances de workflows exécutées dans ce système de gestion de workflow. Pour satisfaire le deuxième objectif, nous avons utilisé les outils CPN [Jen95] pour générer des traces d'exécutions de workflows implantés dans ces outils.

6.3.1 Outil pour la collecte de traces d'exécutions greffé à `Bonita`

Bien que la couche de persistance de `Bonita` reporte et enregistre la liste des projets ayant été exécutés, ces données initialement collectées restent toute fois dépourvues de plusieurs détails qui demeurent importants pour le processus de découverte de workflows. Dans cette section, nous allons présenter une API qu'on a greffée à `Bonita` pour la collecte des traces d'exécutions. Cette API va générer un fichier de traces d'exécutions qui va contenir des informations se rapportant aux événements qui ont eu lieu lors de l'exécution d'instances de workflows. Cet outil permet de collecter tout événement se produisant lors de l'exécution d'une activité faisant partie d'un workflow défini dans `Bonita`.

Parmi les services qu'intègre `Bonita` pour le contrôle des aspects coopératifs se trouve le service de messages JMS qui est utilisé par les composants logiciels de `Bonita` pour assurer un échange d'informations et de messages relatant l'exécution d'un workflow. En effet, il s'agit d'une API de la plate-forme J2EE qui permet de construire des messages pour la transmission des données. Chaque interaction entre l'utilisateur et l'application (telles que, la création d'un projet ou sa suppression, l'exécution d'une activité ou sa terminaison , etc.) sera enregistrée en tant qu'événement JMS.

Ainsi, les services de messagerie JMS permettent de fournir tous les événements qui viennent de se produire lors de l'exécution des activités d'un workflow dans `Bonita`. En nous basant sur cette observation, nous avons construit notre API de collecte de traces d'exécutions qui peut être assimilée à un espion greffé à `Bonita` collectant toutes les transactions qui vont être produites par les services de messageries JMS lors de l'exécution d'une instance de workflow. Chaque événement rapporte le nom du projet, le nom de l'activité exécutée, son état, la date de son exécution, etc. Toutes ces informations seront collectées et regroupées dans un fichier XML de traces d'exécutions relatif au workflow exécuté et contenant toutes ses instances.

Concrètement, les services de messagerie JMS reposent sur un mode publication/abonnement dans des sujets (Topics). Plusieurs clients peuvent envoyer des messages dans ce topic et ce sont les brokers de message qui vont assurer l'acheminement des messages à chaque client qui est préalablement abonné à ce topic. Le message possède donc potentiellement plusieurs destinataires. Dans ce cas, l'émetteur du message ne connaît pas les destinataires qui se sont abonnés.

L'API de collecte qu'on a dénommée *LogListener* (voir schéma UML de la figure C.7 dans le chapitre C) va se mettre en écoute des messages échangés lors de l'exécution d'une instance de workflow, en s'abonnant au topic spécifique. Elle va écouter les messages postés dans ce topic qui se produisent à chaque nouvel événement. Son rôle consiste à extraire ces événements, les transformer en lignes XML, et les enregistrer dans le fichier de traces d'exécutions correspondant à l'instance de workflow exécutée. En effet, à chaque instanciation, le *LogListener* produit un fichier XML relatif à ce cas. Finalement, tous les fichiers relatifs aux cas d'exécution d'un workflow

Figure 6.3 Exemple d'un fichier XML.

seront regroupés dans un fichier unique regroupant les traces d'exécutions de ce workflow. La figure 6.3 représente un fichier XML représentant les traces d'exécution d'une instance d'un workflow dans `Bonita`.

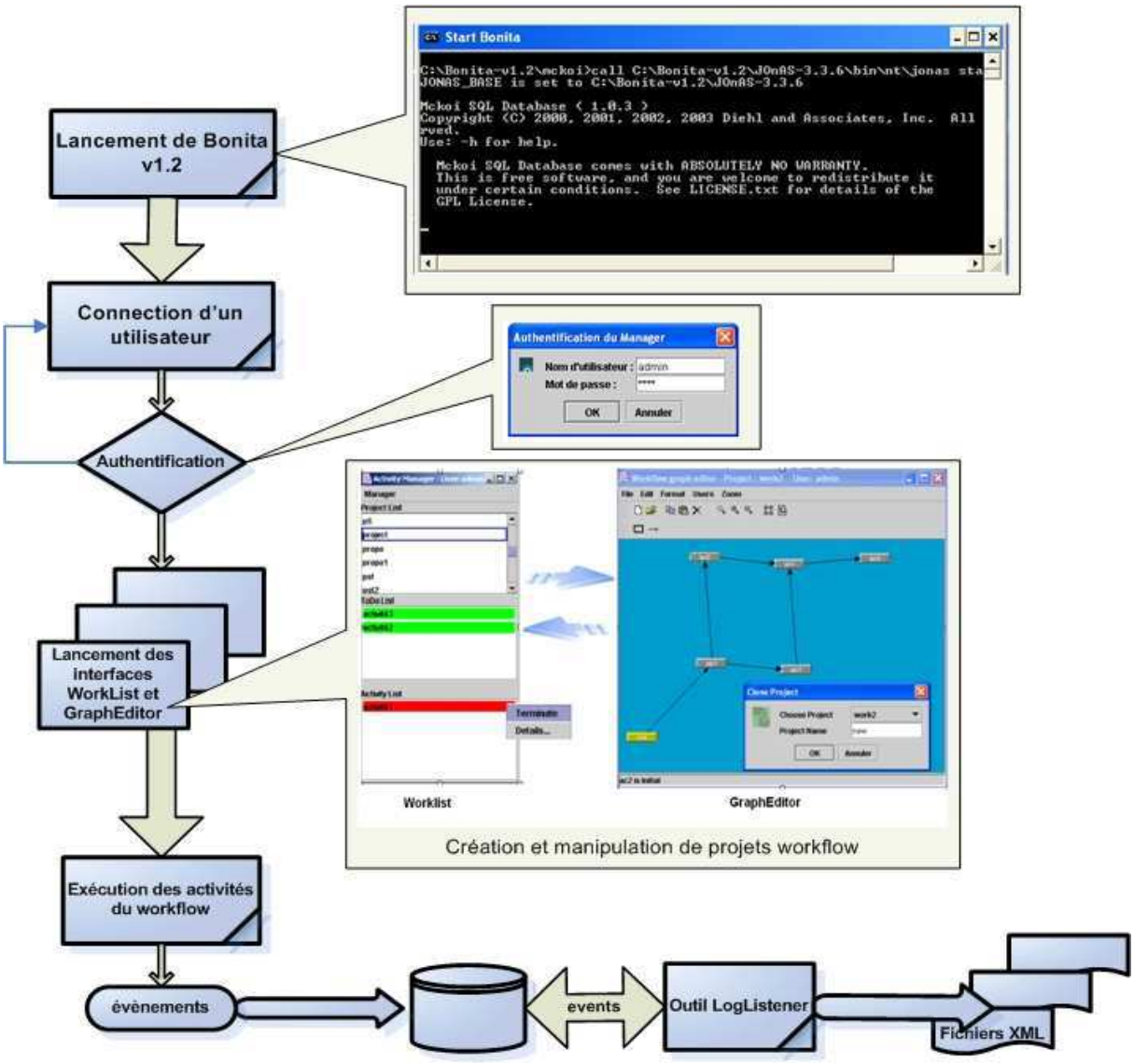
La figure 6.4 illustre un scénario de collecte de traces d'exécutions dans `Bonita`. Une fois `Bonita` lancé, il faut que l'utilisateur soit authentifié. Après l'authentification de l'utilisateur, les interfaces locales (éditeurs graphiques) de `Bonita` sont lancées, c'est là où l'utilisateur peut procéder à l'instanciation du workflow. Grâce à l'interface GraphEditor, l'utilisateur arrive à créer de nouveaux projets workflow ou à manipuler des projets workflow existants. L'execution d'une instance de workflow produira des événements qui sont collectés par le *LogListener* produisant les fichiers XML correspondants.

Une fois les projets de workflow exécutés et leurs traces d'exécutions collectées dans les fichiers XML, nous avons eu recours aux parseur XML JDOM [Har03] comme outil de filtrage et de lecture de données. Le but de JDOM est de faciliter la manipulation au sens large de documents XML (lecture d'un document, représentation sous une forme arborescente, manipulation de cet arborescence, exportation vers des nouvelles structure arborescentes, etc.). En particulier, JDOM effectue une analyse lexicale pour chaque fichier XML généré par l'outil de collecte de traces d'exécutions et le traduit au format XML adopté dans notre approche (*c.f.* section 5.2.2). Il est utilisé comme outil aussi pour parcourir chaque document et extraire des données sur les activités (tels que leur nombre d'occurrence, la liste des activités qui les précèdent et celles qui les suivent, etc.).

6.3.2 Simulation de traces d'exécutions

Pour examiner la capacité de notre méthode à surmonter les difficultés croissantes de passage à l'échelle, nous développons occasionnellement des données simulées. En effet, obtenir des traces d'exécutions réelles pour des exemples de workflow de très grandes tailles s'avère une tâche difficile à l'état actuel. L'avantage d'exécuter des expériences utilisant la simulation est qu'il est plus facile de commander des variables externes, assurant une meilleure validité interne.

Figure 6.4 Scénario de la collecte de traces d'exécutions.



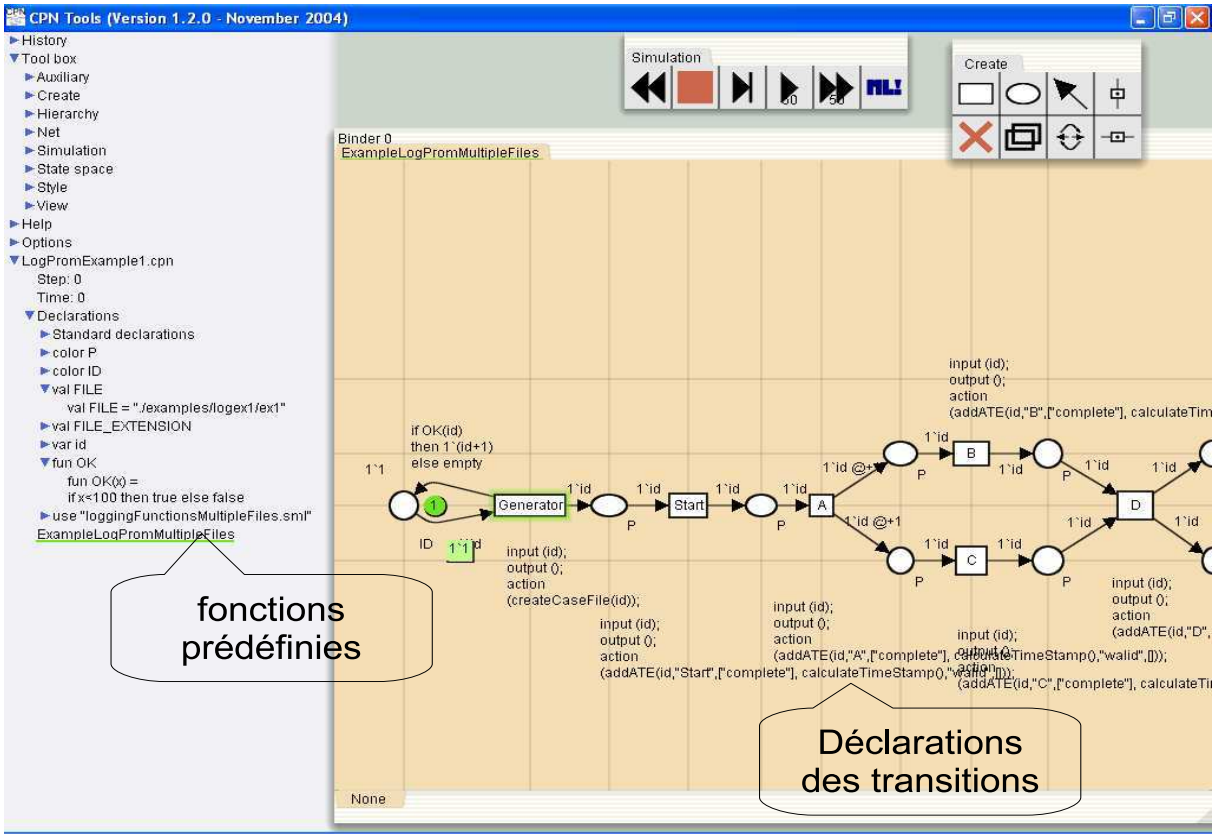
Les modèles sont extérieurement validés en testant et en vérifiant le modèle contre des données résultant de différents exemples de workflow simulés. L'issue du passage à l'échelle de nos tests de validation est mieux gérer par les traces d'exécutions simulées qui nous permettent de couvrir un ensemble très varié qualitativement et quantitativement.

Pour ce faire, nous présentons un outil pour la simulation d'exécution d'instances de workflows pour la génération de traces d'exécutions. L'idée principale est de créer les traces d'exécutions aléatoires XML en simulant des workflows modelés en utilisant les outils CPN[30]. Les outils CPN supportent la modélisation, l'exécution et l'analyse des réseaux de Pétri colorés [Jen95]. Ces outils nous permettent de créer des traces d'exécutions simulées conformes à la structure XML commune proposée dans la section 5.2.2.

[30] http ://wiki.daimi.au.dk/cpntools/cpntools.wiki

Des modifications ont été apportées aux outils CPN pour appeler les fonctions prédéfinies (*c.f.* figure 6.5) qui créeront des traces d'exécutions pour chaque instance exécutée par CPN. Cette étape implique des modifications au niveau de la modélisation des déclarations des workflows CPN, particulièrement au niveau des actions et des entrées/sorties des transitions. Les déclarations sont modifiées pour importer des fonctions pour collecter les traces d'exécutions de chaque transition. Ces fonctions indiquent en particulier l'endroit, le préfixe et l'extension des fichiers XML que CPN crée pour chaque instance qu'il exécute.

Figure 6.5 Les outils de simulation CPN.



Une fois que les déclarations du workflow CPN ont été mises à jour, des déclarations relatives aux transitions (*c.f.* figure 6.5) sont modifiées pour appeler les fonctions de collecte de traces d'exécutions. Le workflow CPN crée un fichier de traces d'exécutions XML pour chaque cas exécuté. Par la suite, les transitions sont collectées une par une après leur exécution. Ces fonctions notent l'exécution d'une transition dans les traces d'exécutions d'un cas. Les paramètres collectés concernent : l'instance de workflow exécutée, le nom, le type d'événement, le temps d'exécution et l'agent responsable de la transition représentant l'activité exécutée.

Par la suite, nous utilisons l'outil ProMimport[31] pour grouper les traces d'exécutions pour les instances d'exécution dans un fichier XML unique qui représente toutes les traces d'exécutions du même workflow. L'exemple de la figure 6.5 est employé pour illustrer comment modifier un workflow CPN pour créer des traces d'exécutions XML pendant son exécution. Les traces

[31] www.promimport.sourceforge.net

d'exécutions de cet exemple sont reprises dans l'annexe D. Dans cet annexe nous allons décrire un ensemble varié d'exemples de workflows simulés.

6.3.3 Synthèse

Dans cette section, nous avons présenté deux solutions logicielles complémentaires pour l'acquisition de traces d'exécutions nécessaires à la validation de nos techniques de découverte. La première solution nous a menée à implanter un outil de collecte de traces d'exécutions réelles au système de gestion de workflows `Bonita`. En effet, lors de l'exécution du workflow, `Bonita` collecte l'ensemble des événements enregistrés traduisant l'exécution réelle du workflow. Une fois les événements collectés, `Bonita` va les retourner sous forme de fichier XML conforme au format adopté dans notre approche.

La deuxième solution est proposée pour satisfaire la complétude du panel d'exemples de workflow nécessaires aux tests de validation de nos techniques de découverte. Cette solution s'inscrit dans une approche de simulation de traces d'exécutions par les outils CPN qui nous permet de mieux diversifier l'ensemble de scénarios de workflows traités.

6.4 Environnement de découverte de workflows

Dans cette section, la faisabilité de notre approche est démontrée par un environnement de découverte de workflow. Cet environnement implémente les algorithmes de découverte que nous avons développés dans notre approche de découverte de patrons de workflow et propose une interface graphique interactive de visualisation. En particulier, notre but est de mettre en œuvre des outils qui vont exploiter l'ensemble des traces d'exécutions réelles ou simulées pour la validation de notre approche. Pour ce faire, nous avons développé un prototype de découverte et de calcul de performance de workflows que nous avons nommé `Workflowminer`. Nous avons œuvré, par la suite, à l'intégration de nos techniques dans la plateforme découverte de workflows `ProM` afin de procéder à des tests de comparaison avec d'autre outils de découverte, et d'accroître la visibilité de notre approche.

6.4.1 `Workflowminer`

6.4.1.1 Introduction

`Workflowminer` est un moteur d'analyse de traces d'exécutions de workflow. Le terme «Miner», qui traduit littéralement «exploitation minière», définit le but du projet, qui consiste à utiliser et exploiter les traces d'exécutions pour découvrir en particulier le graphe du workflow ou analyser les performances du workflow traité. Ce moteur est spécifié en logique de prédicat du 1^{er} ordre et est développé en XProlog[32] sous Java. Il reprend les calculs statistiques et les règles de découverte de patrons de workflow que nous avons spécifiés dans le chapitre précédent. Notons que `Workflowminer` est le produit d'une étroite collaboration inter-universitaires entre ECOO Nancy-france et ENSIAS[33] Rabat-Maroc.

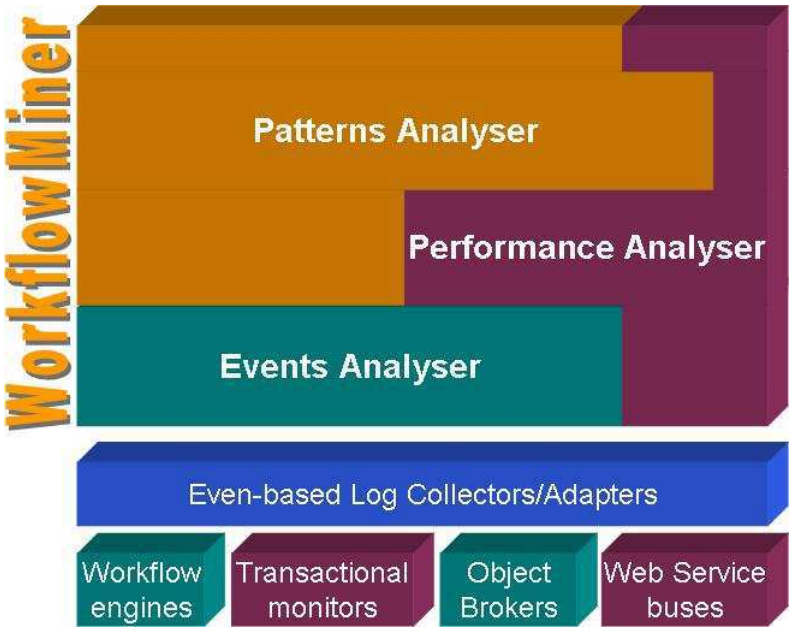
6.4.1.2 Structure

Pour mieux gérer notre projet, nous avons essayé de le découper en modules, tel que chaque module représente un ensemble de tâches relativement indépendantes de l'application. La figure

[32] XProlog, www.iro.umontreal.ca/∼vaucher/XProlog/
[33] École Nationale Supérieure d'Informatique et d'Analyse des Systèmes

6.6 présente l'ensemble des modules du projet ou l'architecture générale de notre application, que l'on peut diviser en quatre parties essentielles :

Figure 6.6 Architecture de `Workflowminer`.



Transformation des événements en prédicats `Prolog` : La couche inférieure «Events-based Log Collectors/Adapters» se charge du chargement des traces d'exécutions, à partir des fichiers XML, et de les adapter au format de prédicats `Prolog` de 1^{er} ordre «Prolog facts». La source des traces d'exécutions peut être n'importe quel système d'information (SGWF, moniteur transactionnel, bus de Web services, etc.) sujet d'un processus de découverte ou d'analyse de performances. Notre but fût de centraliser l'accès à ces données pour ne pas recharger les traces d'exécutions dans chacune des vues de l'application. Pour ce faire, on a construit une API qui se charge du stockage des données en mémoire afin permettre l'accès direct à ces données. On a ainsi formalisé la hiérarchie sur laquelle se base les traces d'exécutions sous forme d'objets. Cette API présente plusieurs avantages au niveau de la persistance et la non redondance des données ainsi qu'au niveau de l'amélioration des performances et du temps du traitement des données.

Conception du moteur d'analyse de traces d'exécutions : La couche «Event Analyser» définit un moteur d'analyse de ces «Prolog facts» et découvre, à travers notre technique d'analyse statistique, les dépendances causales entre les événements des traces d'exécutions données par la couche précédente. Ce module constitue le cœur de l'application car il fournit les différentes dépendances entre les événements qui serviront par la suite à la construction de la table de dépendances et enfin la mise en place du graphe de workflow.

Découverte des patrons de workflows : La couche «Patterns Analyser» utilise les règles que nous avons définies dans le chapitre précédent pour la découverte des patrons de workflow (*c.f.* section 5.3.4). Ces règles sont reproduites sous la forme de prédicats de 1^{er} ordre et passées comme entrée avec les statistiques fournies par «LogAnalyser» au moteur `Prolog`. Le moteur `Prolog` les analyse pour découvrir les patrons de workflow correspondants.

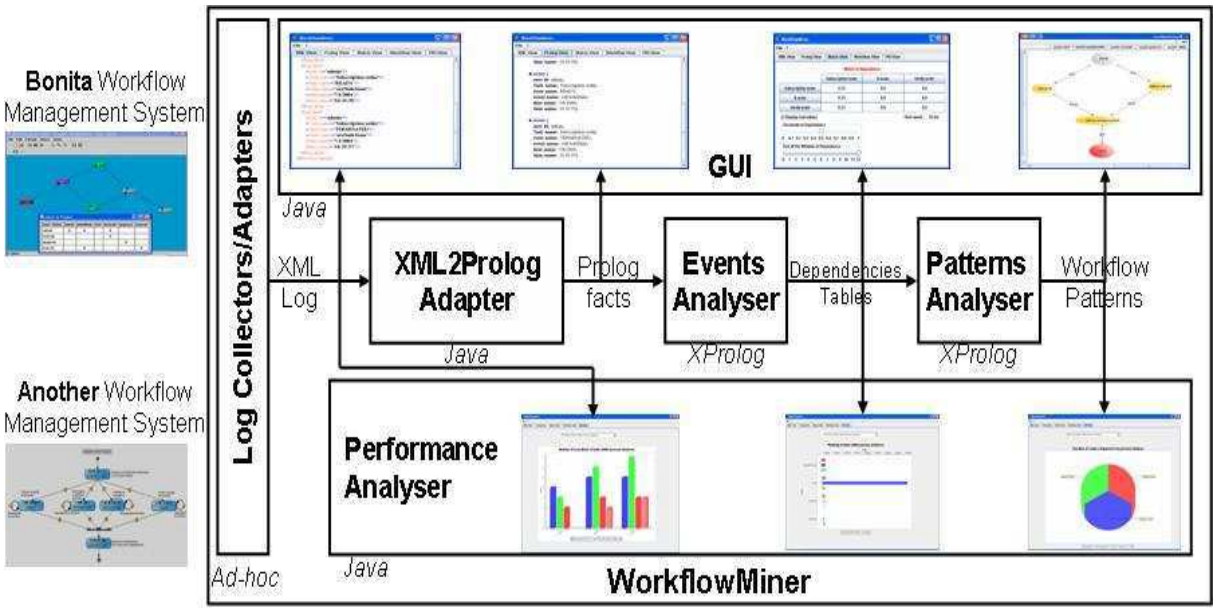
Calcul de performances : La couche «Performance Analyser» utilise les dépendances causales et les patrons de workflow découverts pour mesurer des performances métriques du workflow. Elle contient un composant tableau de bord pour mesurer la performance des workflows,

moyennant des indicateurs clefs de performance (KPI ou «Key Performance Indicators»). Ces indicateurs de performance mesurent la performance d'un workflow dans une vision transverse des instances de ses activités. Cette dernière couche «Performance Analyser» est un composant additionnel qui ne fait pas vraiment partie de l'approche de découverte de workflow mais qui entre dans notre démarche de re-ingénierie de procédés.

6.4.1.3 Affichage

Pour pouvoir visualiser le résultat de la découverte (*c.f.* figure 6.7), on a conçu une interface entre l'utilisateur et l'application, qui permet la visualisation du résultat sous forme d'un graphe de workflow. Nous avons fait de sorte à ce qu'on puisse appliquer notre processus de découverte pas à pas. L'intérêt d'une telle interface est de fournir un outil qui sort de la simple perspective de présentation du résultat de découverte à une perspective d'interaction avec l'utilisateur via une interface intuitive. Pour ce faire, notre application offre une interface graphique conviviale contenant quatre onglets tel que chaque onglet représente une vue de chacune des perspectives décrites ci-dessus.

Figure 6.7 Interface graphique de `Workflowminer`.



Le premier onglet «XML VIEW» permet de visualiser automatiquement les traces d'exécutions sous forme d'un schéma XML. Le second onglet «`PROLOG` VIEW», affiche les événements `Prolog`, extraits du fichier XML source. Les événements sont de la même structure que les tâches XML mais sous forme d'événements `Prolog`. Ceci est réalisé grâce au processeur XSLT qui transforme le code de la structure XML en un code HTML affichable par notre application. Le volet «MATRIX VIEW» permet l'affichage de la matrice déduite toujours du fichier source XML. Chaque élément représente la probabilité entre deux nœuds. Cette étape pourrait se résumer en une simple traduction de la matrice de dépendances sous forme de graphe directionnel. Les nœuds de ce graphe représentent les différentes activités du workflow, et contiennent leurs fréquences d'apparition, alors que les arcs représentent les dépendances entre les activités et contiennent les

informations les concernant. Cet onglet offre à l'utilisateur la possibilité de modifier/visualiser la taille de la fenêtre de concurrence. Pour une meilleure gestion du workflow, nous avons ajouté une *checkbox* «Display initial values» qui permet l'affichage des valeurs initiales avant l'application de la fenêtre de concurrence. Et enfin, le quatrième onglet «Patterns VIEW» permet la régénération et l'affichage du nouveau workflow. Cette vue nous permet de visualiser les patrons de workflows découverts composant le schéma de workflow global. La représentation graphique de ces patrons permet de les isoler et de les étudier un par un.

6.4.2 Plug-in «`Workflow patterns miner`» dans `ProM`

Les outils de découverte utilisent généralement différents formats pour collecter ou lire les traces d'exécutions et présentent leurs résultats dans différents modèles de procédés. Ceci rend difficile la (re)utilisation de différents outils pour le même ensemble de traces d'exécutions et la comparaison des résultats de découverte. Par ailleurs, certains de ces outils mettent en application des concepts qui peuvent être très utiles pour d'autres outils, mais il reste difficile de les combiner et les réutiliser dans des environnements indépendants. En conséquence, les chercheurs travaillant sur des nouvelles techniques de découverte se retrouvent à construire une infrastructure de découverte à partir de zéro et à examiner leurs techniques d'une façon isolée, se déconnectant des applications courantes. Pour répondre à ces différentes issues, nous avons choisi d'implanter notre approche de découverte comme un plug-in nommé «`Workflow patterns miner`» dans la plateforme `ProM` qui fédère un grand nombre d'outils et de techniques de découverte de procédés et permet à ces plug-ins de coopérer d'une manière normalisée. Notons que le développement de ce plug-in s'inscrit dans le cadre d'une collaboration inter-universitaires entre ECOO Nancy-france et TU/e[34] Eindhoven-Hollande, supportée par le projet européen NoE-Interop[35].

`ProM` est une plateforme extensible qui supporte une grande variété de techniques de découverte de procédé sous forme de plug-ins. `ProM` offre une certaine flexibilité sur le format d'entrée et de sortie grâce à un ensemble d'outils de conversion. Il est également assez ouvert pour tenir compte d'une réutilisation facile du code pour l'implantation de nouvelles technique de découverte de procédé. Un autre dispositif important dans la plateforme `ProM` est qu'elle permet l'interaction entre un grand nombre de plug-ins de traitement de traces d'exécutions, de conversion de modèles de procédés, de découverte de workflow, etc.

`ProM` est implanté en Java et contient plus de 90 plug-ins disponibles traitant de la découverte de procédés, la vérification des modèles de procédés, la vérification des formules linéaires logiques, la conformance entre le modèle de procédés et ses traces d'exécutions, et l'analyse de performances. `ProM` supporte aussi l'importation et la conversion de plusieurs langages de modélisation de procédé tels que : les réseaux de Petri (PNML, TPN), les graphes de transitions états, YAWL, etc. Les plug-ins de découverte couvrent la découverte de flot de contrôle (Alpha algorithme, découverte génétique, découverte multi-phase, ...), l'analyse de la perspective organisationnelle (Social Network miner, the Staff Assignment miner, ...) et la perspective données (the Decision miner, ...). `ProM` est libre source, sous une licence publique commune, et peut être téléchargé gratuitement[36].

Comme nous l'avons indiqué, la base pour toutes les techniques de découverte de procédé est les traces d'exécutions. `ProM` a développé un format générique XML de taces d'exécutions. Ce format se base sur une comparaison complète des besoins d'entrée de divers outils de découverte existants et les traces d'exécutions typiquement produites par les workflows. Il supporte un grand

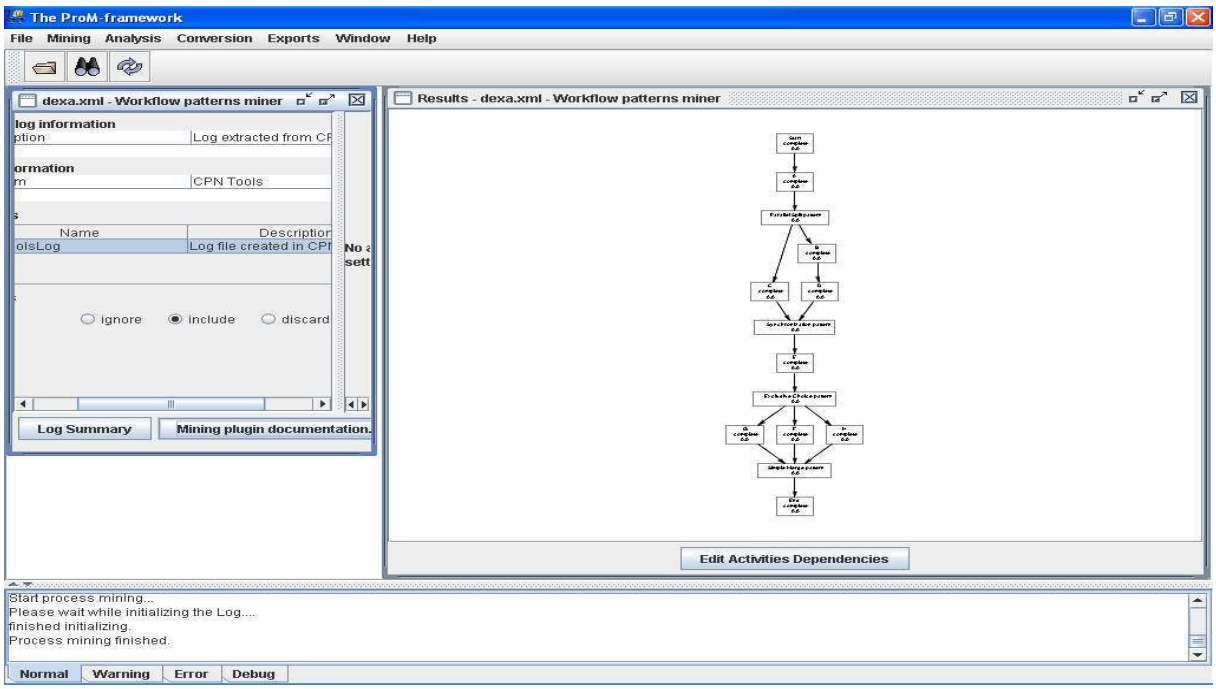
[34] Technische Universiteit Eindhoven
[35] www.interop-noe.org
[36] www.processmining.org

nombre de filtres de traces d'exécutions, qui sont un outil nécessaire pour une structure de traces d'exécutions commune. Ce composant peut traiter une grande variété de traces d'exécutions.

ProM est un environnement complètement «pluggable». L'ajout de plug-ins dans ProM peut être fait avec une facilité relative. Une fois le plug-in prêt et archivé dans un «.jar» spécifique, il peut être ajouté à la plateforme en modifiant principalement quelques lignes aux fichiers de configuration (fichiers d'extension «.ini»). Il n'y a aucun besoin de modifier la plateforme ProM. Ainsi, aucun changement du code de la plateforme principale n'est nécessaire, c.-à-d., de nouveaux plug-ins de découverte peuvent être ajoutés sans recompiltation.

Pour l'implantation de notre plug-in «`Workflow patterns miner`», nous avons pu profiter de plusieurs plug-ins et bibliothèques de la plateforme ProM. Essentiellement, le chargement, la lecture, la conversion et le traitement des fichiers de traces d'exécutions sont pris en charge par des APIs génériques de ProM. ProM offre aussi un large panel d'outils d'analyse des traces d'exécutions qui a pleinement simplifié l'implantation de nos algorithmes d'analyse statistique des traces d'exécutions nécessaires à notre approche de découverte de patrons de workflows.

Figure 6.8 Le plug-in «`Workflow patterns miner`» dans ProM.

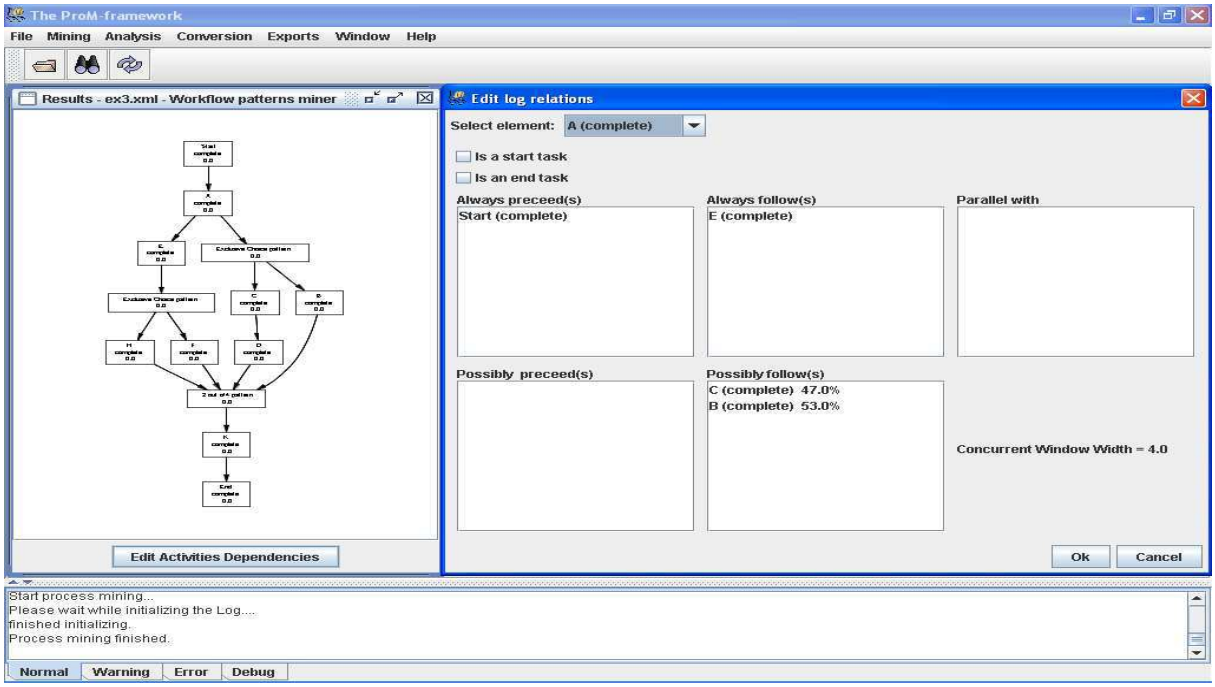


La figure 6.8 montre une capture d'écran de notre plug-in «`Workflow patterns miner`» dans ProM. Nous pouvons observer que l'aspect graphique du plug-in suit l'ergonomie de la plateforme. En effet, ProM recommande un panel graphique pour la visualisation des entrées/sorties du plug-in. Correctement, au lancement de ProM l'utilisateur fera le choix entre les différents types de plug-ins existants. Après le choix des plug-ins de découverte, l'utilisateur fera le choix du fichier de traces d'exécutions à traiter. Notons que cette partie est prise en charge par la plateforme et est identique à tous les plug-ins. Au lancement du processus de découverte, le plug-in entre en jeu et prend la main dans un thread différent.

Dans la figure 6.8, nous pouvons observer en particulier deux frames de notre application : une qui concerne les traces d'exécutions et l'autre la visualisation du workflow découvert. La première

frame sert à fixer un certain nombre d'options sur la structure de traces d'exécutions pour définir la nature de la structure par le choix des événements traités. Notre choix sera porté sur une structure simplifiée (*c.f.* section 5.2.3), qui se limite au traitement de l'événement **terminé** dans ces traces, si le processus de découverte concerne le flot de contrôle. Par contre, le choix d'une structure de traces enrichie est nécessaire pour la découverte du comportement transactionnel. Le deuxième frame concerne le schéma de workflow découvert composé par l'ensemble des patrons qui le constituent. Nous avons utilisé la bibliothèque graphique de visualisation de graphes nommée «`Dot`» [Gra] pour la construction du graphe de workflow.

Figure 6.9 Table de dépendances dans «`Workflow patterns miner`».



Il faut également préciser que nous avons entamé un travail d'implantation pour améliorer l'aspect visuel de notre plug-in pour répondre au besoin d'interaction avec l'utilisateur. Ce dernier point est important parce que la re-ingénierie de procédés est souvent un processus interactif où l'interprétation humaine est une connaissance additionnelle importante et peut être employée pour améliorer l'exploitation du résultat du processus de découverte. Pour ce faire, nous avons ajouté un bouton dans le deuxième frame de découverte pour la visualisation de différentes informations sur chaque activité concernant la fréquence des activités qui la précédent ou qui la suivent, les activités qui lui sont parallèles et la taille de sa fenêtre de concurrence (*c.f.* figure 6.9).

L'implantation du plug-in «`Workflow patterns miner`» dans la plateforme `ProM`, nous a permit de mener une étude comparative avec les autres techniques de découverte qui sont dans leur majorité implantés dans `ProM`. Cette étude comparative a été la base du tableau comparatif que nous avons présenté dans le chapitre précédent (*c.f.* tableau 5.8). Notons que notre comparaison était d'ordre qualificatif, ne traitant que des aspects concernant certaines qualités et caractéristiques des entrées/sorties. Une étude comparative d'ordre quantitative qui concerne la rapidité et la complexité des algorithmes de découverte est erronée puisque les algorithmes de découverte

ne possèdent pas généralement les mêmes entrées/sorties qui sont une caractéristique nécessaire à une étude comparative quantitative.

6.4.3 Synthèse

Dans cette section, nous avons présenté deux solutions logicielles que nous avons développées pour la validation de nos techniques de découverte de patrons de workflows. La première solution consiste au développement du prototype `Workflowminer` qui se base sur des prédicats de `Prolog` pour l'analyse des traces d'exécutions et la découverte de patrons de workflows. Ce prototype propose aussi un composant d'analyse de performances. Nous avons aussi implanté le plug-in «`Workflow patterns miner`» dans la plate forme `ProM`. Ce plug-in a été sujet d'un ensemble de tests de validation dans l'annexe D, où un panel d'exemples de workflows ont été simulés et leurs traces d'exécutions récupérées comme entrée test pour notre plug-in.

6.5 Conclusion

Dans ce chapitre, nous avons abordé plusieurs aspects pour la validation de notre approche de modélisation et de découverte de workflows transactionnels. Nous avons proposé, dans le cadre d'un outil de modélisation de workflow transactionnel, une extension du système de gestion de workflow `Bonita` pour l'intégration du comportement transactionnel. Par la suite, nous avons proposé des outils pour la validation de nos techniques de découverte de workflows. Ainsi, nous avons présenté des outils pour la collecte de traces d'exécutions simulées ou réelles. Et, nous avons implanté notre algorithme de découverte de patrons de workflows dans deux environnements différents : le prototype `Workflowminer` qui est un environnement indépendant et autonome et le plug-in «`Workflow patterns miner`» dans la plateforme commune `ProM` de découverte de workflows.

Néanmoins, notre approche pour la re-ingénierie de procédés par l'analyse des traces d'exécution nécessite un effort d'implémentation supplémentaire pour intégrer particulièrement les différentes règles d'amélioration et de correction que nous avons proposées dans le chapitre précédent. Ainsi nous visons l'implantation d'un plug-in dont le rôle sera d'exploiter les résultats de découverte pour proposer des suggestions d'amélioration et de correction aux concepteurs d'une manière interactive. Cependant le défis majeur auquel nous faisons face afin de répondre à cet objectif est l'acquisition de cas d'utilisation réels contenant des anomalies conceptuelles causant des échecs d'exécutions variés. En effet, l'exploitation de traces d'exécution simulées pour la validation de ce plug-in est loin d'être suffisante pour tirer les conséquences et percevoir les avantages de l'utilisation notre approche de re-ingénierie de procédés par l'analyse des traces d'exécutions.

Chapitre 7

Bilan et perspectives

L'objectif de ce travail de thèse était d'assurer **une (re)conception continue** par l'analyse des traces d'exécutions qui répond aux problèmes de **fiabilité et d'évolution** des procédés métiers. Concrètement, nous nous sommes intéressés à développer une approche de fiabilisation des exécutions de procédés métiers à travers une approche originale de découverte de workflow. Dans ce chapitre, nous voulons résumer notre travail de recherche par la présentation de notre contribution, sa motivation, et son originalité (*c.f.* section 7.1). Ensuite, nous présentons quelques perspectives à nos travaux de recherche (*c.f.* section 7.2).

7.1 Travail réalisé et contributions

Les approches actuelles de modélisation de procédés métiers se sont principalement inspirées des (i) MTA en ce qui concerne la coordination transactionnelle et (ii) de l'approche traditionnelle «workflow» en ce qui l'ordonnancement des tâches. Or ces deux approches présentent des limites pour intégrer les trois dimensions de «flexibilité», de «correction» et d'«adéquation aux procédés métiers». D'une part, les MTA assurent un bon niveau de correction au détriment des dimensions «flexibilité» et «procédés métiers». D'autre part, les systèmes de workflow classiques assurent un bon niveau de flexibilité et s'adaptent bien aux contextes de «procédés métiers». Cependant, ils ignorent relativement la dimension de «correction».

Ayant ces trois dimensions comme principes directeurs, nous avons proposé un modèle de workflow transactionnel (WfT) qui permet (i) d'enrichir la description du modèle de workflow pour mieux exprimer les propriétés transactionnelles des activités et les dépendances transactionnelles entre les activités et (ii) de modéliser des mécanismes de recouvrement en cas d'échec d'exécution pour assurer des exécutions fiables. Nous avons distingué en particulier entre le `flot de contrôle` et le `flot transactionnel` d'un workflow transactionnel. Le `flot de contrôle`, qui est décrit à travers les patrons de workflow, définit l'ordre d'invocation des activités composantes. Le `flot transactionnel`, qui est décrit à travers les propriétés transactionnelles des activités et les dépendances transactionnelles entre les activités, définit les mécanismes de recouvrement en cas d'échecs.

L'originalité de notre modèle de workflow transactionnel est qu'il a pu réconcilier les systèmes de workflow et les MTA. D'une part, un WfT peut être considéré comme un modèle de workflow classique qui reprend les patrons de workflow les plus communs. D'autre part, il peut être considéré aussi comme une transaction structurée où les activités composantes sont les sous-transactions et les interactions sont les dépendances transactionnelles. Par rapport au modèle de workflow classique, notre modèle permet d'intégrer le `flot transactionnel`, un concept qui

a été (relativement) ignoré. Par comparaison aux MTA, notre modèle offre plus de flexibilité et d'adéquation aux procédés métiers.

Par ailleurs, les approches classiques de conception de procédés prennent généralement comme point de départ les perceptions et les suppositions des concepteurs et souffrent d'un manque d'implication des utilisateurs tant dans la conception que la vérification. En plus, elles manquent de mécanismes de correction permettant d'optimiser la structure du workflow initialement conçue du fait qu'elle analyse un modèle figé et ne traite pas du besoin d'évolution pendant la phase d'exécution. Afin de répondre au besoin d'évolution et d'adaptation du schéma du workflow suite à des nouvelles contraintes d'exécution ou à des besoins d'évolution exprimés par l'utilisateur après l'établissement du procédé, nous avons spécifié une nouvelle étape d'amélioration et/ou de correction du comportement transactionnel du workflow par l'analyse des traces d'exécutions. Une analyse statistique des traces d'exécutions à été être effectuée pour fournir l'entrée pour une nouvelle phase de (re)conception dans le cadre d'une re-ingénierie du procédé.

Concrètement, nous avons utilisé les traces d'exécutions de workflows dans une approche originale de découverte de workflow transactionnels qui dérive le modèle de workflow «effectif» avec aussi peu d'information que possible. Notre approche de découverte de workflow transactionnel se rapporte à des méthodes d'analyse statistique des traces d'exécutions. Nous avons commencé par la découverte du `flot de contrôle` en décrivant un algorithme dynamique qui bâtit le résultat final (le workflow final) à partir de résultats locaux (les patrons de workflow). Par la suite, nous avons découvert le `flot transactionnel` du workflow en capturant les dépendances transactionnelles intra-activité et inter-activités décrivant respectivement les propriétés transactionnelles des activités et les dépendances transactionnelles entre les activités.

Les résultats découverts sont utilisés par la suite pour améliorer et/ou corriger les mécanismes de recouvrement du schéma de workflow initialement conçu. La comparaison entre le modèle de workflow «a posteriori» (initialement conçu) au modèle «a priori» (découvert) nous a permis de procéder à une Delta analyse relevant les disparités entre la conception bâtie dans la phase de conception et l'exécution réelle enregistrée dans la phase d'exécution. Ces disparités peuvent être synonyme d'un besoin d'évolution du schéma conceptuel du workflow. En nous basant sur ces disparités, nous avons proposé un ensemble de suggestions corrigeant et améliorant le schéma du workflow pour coller au mieux à la «réalité» de la phase d'exécution. Ces suggestions respectent les caractéristiques sémantiques de notre modèle de workflow transactionnel. De plus, elles tiennent compte des besoins spécifiques des concepteurs. En effet, les corrections induites par ces suggestions ne sont pas faites d'une façon automatique mais plutôt interactive avec les concepteurs pour qu'ils puissent préciser leurs besoins spécifiques.

L'avantage majeur de cette approche, qui est d'une base théorique solide, est sa capacité de coller au mieux à la réalité exprimée dans la phase d'exécution en reproduisant le modèle découvert comme base pour une nouvelle phase de (re)conception. Ainsi, nous pouvons détecter et corriger les erreurs de la phase antérieure de conception et satisfaire les nouveaux besoins d'évolution du procédé. En effet, notre approche comporte une méthodologie conceptuelle originale qui repose sur des faits réels et non sur des hypothèses qui peuvent ne pas coïncider avec la réalité. Elle facilite ainsi le diagnostic du procédé, dans le sens qu'en utilisant des cas réels les concepteurs peuvent proposer des corrections ou/et améliorations du comportement transactionnel du workflow pour un meilleur «alignement» avec la réalité.

Nous nous sommes attelés à mettre en œuvre et à valider nos propositions à travers le développement et l'utilisation d'un certain nombre d'outils et d'applications. D'une part, nous avons étendu le système de gestion de workflows `Bonita` pour supporter notre modèle de workflow transactionnel. D'autre part, nous nous sommes intéressés à la la validation de nos techniques de découverte de workflows. Ainsi, nous avons présenté et développé des outils pour la collecte de traces

d'exécutions simulées ou réelles. Finalement, nous avons implanté notre algorithme de découverte de patrons de workflow dans deux environnements différents : le prototype `Workflowminer` et le plug-in «`Workflow patterns miner`».

7.2 Perspectives

Actuellement, nous entrevoyons trois perspectives au travail présenté. Une première direction concerne l'extension de notre technique de découverte de procédé. Nous distinguons à ce niveau entre deux sous objectifs. Le premier consiste à pouvoir supporter des structures de contrôle plus complexes et gérer des propriétés transactionnelles liées au flux de données. Ceci implique principalement de pouvoir gérer des traces d'exécutions sémantiquement enrichies. Particulièrement, nous pouvons envisager d'ajouter de l'information sur le flux de donnée ou le contexte métier du procédé (par exemple, les raisons de l'évolution). Nous pouvons souligner ici que l'information issue du flux de données nous sera très utile pour une meilleure découverte du comportement transactionnel et ainsi une meilleure connaissance sur les mécanismes de recouvrement et particulièrement la localisation des points cohérents. En effet, l'évolution du procédé est souvent un mélange d'adaptations du flux de contrôle et du flux de données dans un souci de préserver la correction du procédé. Par conséquent, plusieurs issues surgissent, comme par exemple, comment recueillir cette information, comment la représenter dans les traces d'exécutions, comment la rendre efficacement accessible, quelles nouvelles métriques d'analyse ou techniques de fouille utiliser pour s'adapter à la nouvelle structure de traces d'exécutions ou pour améliorer le résultat de découverte, etc.

Nous pouvons aussi, dans le cadre de la même perspective (c.à.d. la découverte de procédé), viser un deuxième sous objectif en s'attelant à traiter des cas d'utilisation concrets. Le but est d'utiliser les techniques de découverte de workflow pour répondre à des besoins métiers plus spécifiques de certains cas d'utilisation. Ainsi, nous envisageons la réalisation de tests d'exécution et de validation dans le système de gestion de workflows `Bonita` sur des scénarios de workflows qui décrivent des cas d'utilisation «industriels» qui peuvent concerner les secteurs hospitaliers ou éducatifs pour découvrir d'autres caractéristiques non fonctionnelles du procédé. Par ailleurs, notre travail de re-ingénierie de procédés par l'analyse des traces d'exécutions nécessite un effort d'implémentation supplémentaire pour intégrer particulièrement les différentes règles d'amélioration et de correction que nous avons proposées. Cependant l'acquisition de cas d'utilisation réels contenant des anomalies conceptuelles dues à des échecs d'exécution reste une tâche ardue à cause de la réticence de certains partenaires industrielles à fournir ce types de traces d'exécutions.

Dans la deuxième perspective, nous partons de la limitation, que nous nous sommes fixée dans notre approche de découverte de procédé, d'avoir les traces d'exécution comme seule et unique entrée. Nous distinguons à ce niveau entre deux sous objectifs. Tout d'abord, il peut ne pas y avoir assez d'événements pour découvrir réellement le modèle de procédé. En conséquence, le modèle découvert peut être défectueux ou non représentatif. En second lieu, la Delta analyse ne fournit pas de mesures quantitatives pour mesurer la «correspondance» entre, d'une part, le workflow initiale et le workflow découvert, et d'autre part, les traces d'exécutions. Ainsi, nous nous proposons dans nos travaux futurs de mesurer cette «correspondance» et de localiser toute déviation possible. En effet, des déviations rares se produiront toujours entre le modèle initialement conçu et le modèle découvert, mais ne devraient pas être considérées comme symptôme d'un besoin d'évolution. Seulement une quantité suffisante d'instances de procédés montrant une déviation récurrente, peut justifier une nouvelle étape de correction/amélioration pour aligner le procédé aux nouvelles contraintes observées.

Par ailleurs, le deuxième sous objectif vise à combiner les données de traces d'exécutions quantitatives avec des données qualitatives recueillies d'autres sources (par exemple, des entrevues et des questionnaires). Cette combinaison peut aider à améliorer notre connaissance et notre compréhension du procédé métier.

Enfin, la troisième perspective vise à adapter/combiner notre approche à/avec des domaines connexes aux services Web. Nous distinguons à ce niveau entre deux sous objectifs. Le premier consiste à appliquer les techniques de découverte de procédé aux compositions de services Web. Un premier travail a été validé dans [BGG06]. Dans ce travail, nous avons appliqué les techniques de découverte décrites dans notre thèse pour découvrir et améliorer le comportement transactionnel d'un service Web composite. Par ailleurs, un des défis majeurs auxquels nous avons dû faire face est la collecte de traces d'exécutions de services Web. Pour ce faire, nous avons mené un travail que nous venons de publier [RGvdA+06]. Dans ce travail, nous proposons une première solution de collecte de traces d'exécution qui se base sur l'écoute des message SOAP [W3C00] échangées. Ces traces d'exécution sont par la suite exploitées pour la découverte et la vérification de certaines propriétés non fonctionnelles des services Web.

Dans le deuxième sous objectif, notre ambition est de situer notre travail concernant l'approche transactionnelle par rapport à la thématique de recherche des services Web sémantiques. Les services Web sémantiques se situent à la convergence de deux domaines de recherche importants qui concernent les technologies de l'Internet, à savoir le Web sémantique et les services Web [KT03]. L'objectif ultime est d'automatiser les processus de découverte et de composition de services Web. L'idée de base est d'enrichir les technologies actuelles par une couche sémantique afin de permettre la manipulation automatique des services Web. Notre approche peut être appliquée pour assurer des compositions fiables d'une façon automatique. S'inscrivant dans la même perspective que [FDDB05; DFDB05], la première étape consiste à enrichir la description de services Web par leurs propriétés transactionnelles. Ceci est faisable dans OWL-S [Coa03] en précisant les propriétés transactionnelles d'un service Web comme des attributs non fonctionnels dans son profil. Ensuite, nous pouvons appliquer nos techniques pour assurer des compositions fiables. Une autre application envisageable de notre approche est d'étendre les constructeurs de contrôle du modèle de service de OWL-S avec des patrons transactionnels [BPG06].

Annexe A

Calcul événementiel

Nous présentons dans cet annexe les principaux prédicats du calcul événementiel et les axiomes nécessaires à leurs interprétations.

A.1 Prédicats

Les principaux prédicats du calcul événementiel sont décrits dans le table A.1.

A.2 Axiomes

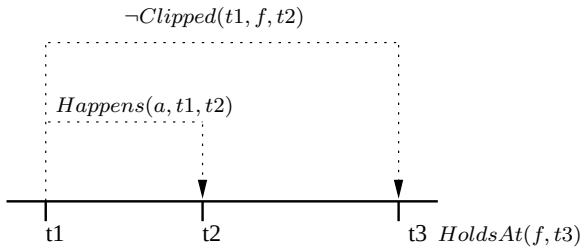
En se basant sur ces prédicats, les axiomes suivants sont identifiés :

1. $HoldsAt(f,t) \longleftarrow Initially_P(f) \wedge \neg Clipped(0,f,t)$
   Toutes les états qui s'intialisent et ne sont pas terminés par aucun événement depuis l'instant 0 à t continuent à persister en t.
2. $HoldsAt(f,t3) \longleftarrow Happens(a,t1,t2) \wedge Initiates(a,f,t1) \wedge (t1 < t2 < t3) \wedge \neg Clipped(t1,f,t3)$
   Comme l'indique la figure 4.1, après qu'un événement initialise un état, cet état persiste tant qu'aucun autre événement ne l'annule.

Prédicat	**Interprétation**
$happens(a,t1,t2)$ $happens(a,t)$	Le déclenchement de l'action a s'est produit entre les instants $t1$ et $t2$ Prédicat simplifié de $happens(a,t1,t2)$ qui se produit à t
$holdsAt(f,t)$	L'état f est vrai à à l'instant t
$initiates(a,f,t)$	L'état f devient vrai après l'action a à l'instant t
$terminates(a,f,t)$	L'état f devient faux après l'action e à l'instant t
$Initially_P(f)$	L'état f est vrai à partir de l'instant initial, c-à-d 0
$Initially_N(f)$	L'état f est faux à partir de l'instant initial, c-à-d 0
$Clipped(t1,f,t2)$	L'état f est terminé entre l'instant $t1$ et $t2$
$Declipped(t1,f,t2)$	L'état f est initialisé à vrai entre l'instant $t1$ et $t2$
$t1 < t2$	relation d'ordre standard pour le temps, $t1$ précède $t2$

TAB. A.1 – Prédicats du calcul événementiel

Figure A.1 Axiome 2

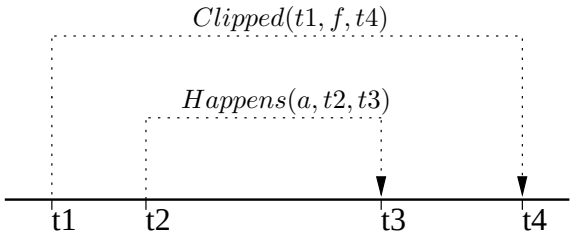


Domaine de description: $Initiates(a, f, t1)$

3. $Clipped(t1, f, t4) \longleftrightarrow \exists a, t2, t3[Happens(a, t2, t3) \wedge (t1 < t2) \wedge (t2 < t3) \wedge (t3 < t4) \wedge Terminates(a, f, t2)]$

   Comme l'indique la figure A.4, cet axiome stipule qu'un état est dit «clipped» si et seulement si un événement se produit pour l'annuler.

Figure A.2 Axiome 3



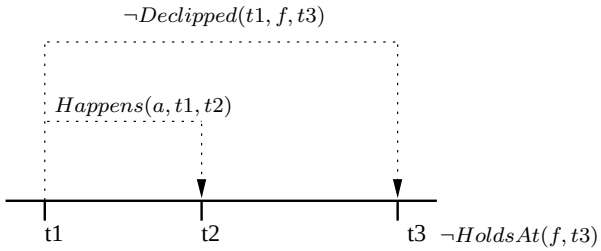
Domaine de description : $Terminates(a, f, t2)$

4. $\neg HoldsAt(f, t) \longleftarrow Initially_P(f) \wedge \neg Declipped(0, f, t)$

   Cet axiome montre qu'un état est annulé s'il n'est pas initialement persistant et qu'il n'y a aucun événement qui l'a déclenché.

5. $\neg HoldsAt(f, t3) \longleftarrow Happens(a, t1, t2) \wedge Terminates(a, f, t1) \wedge (t1 < t2) \wedge (t2 < t3) \wedge \neg Declipped(t1, f, t3)$ Comme le montre la figure 4.3, si un événement se produit et termine un état, et aucun autre événement ne se produit pour l'initialiser, alors cet état ne persiste plus.

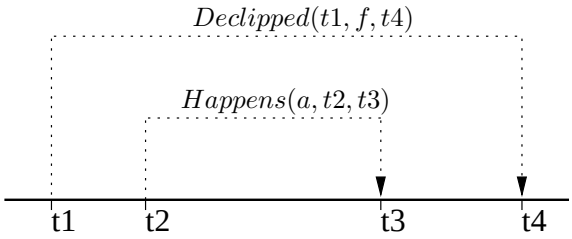
Figure A.3 Axiome 5



Domaine de description: $Terminates(a, f, t1)$

6. $Declipped(t1, f, t4) \longleftrightarrow \exists a, t2, t3\ [Happens(a, t2, t3) \wedge (t1 < t2) \wedge (t2 < t4) \wedge (t3 < t4) \wedge Initiates(a, f, t2)]$

Figure A.4 Axiome 6



Domaine de description : $Initiates(a, f, t2)$

Un état est dite «declipped» dans une période si et seulement il existe un événement qui se produit et initialise ou réalise l'état dans cette période. Cet axiome est illustré par la figure 4.4.

Les axiomes 1 et 3 indiquent qu'un état est vrai en un temps t si et seulement si soit il était initialement vrai et qu'il n'est pas terminé entre 0 et t par une action, soit qu'une action, ayant pour effet d'initier l'état, se produise avant t et qu'entre cet instant et t l'état n'est pas terminé par une autre action. Cela est repris par la formule suivante. L'axiome 6 spécifie qu'un état se termine entre t1 et t3, si et seulement si une action se produit dans l'intervalle]t1 ; t3[, et cette action, notée a, a pour effet de terminer cet état.

Annexe B

Démonstrations des théorèmes et des lemmes

Dans cet annexe, nous donnons les démonstrations des théorèmes et des lemmes que nous avons introduit dans chapitre 5

B.1 Démonstration du lemme 5.1 sur le nombre d'instances pour des traces d'exécutions complètes

Énoncé du lemme 5.1
soit wft un workflow transactionnel le nombre minimal et suffisant de traces d'exécutions d'instances différentes pour la découverte du `flot de contrôle` wft est défini comme suit :

1. Le nombre est égal à 1 pour les workflow ne contenant qu'une suite séquentielle d'activités sans comportement concurrentiel ni conditionnel ;
2. Un comportement conditionnel entre n activités avant un point de «jointure» ou après un point de «diffusion» nécessite n traces d'exécutions d'instances différentes reprenant à chaque fois un choix.
3. Un comportement concurrentiel entre n flux de contrôle, chaque flux i ; $0 < i < n+1$ contient j_i activités nécessite $\Pi_{i=1..n}(j_i{+}i{-}1) = j_i{*}(j_2{+}1){*}(j_3{+}2){*}(j_4{+}3){*}.....{*}(j_5{+}n{-}1)$ traces d'exécutions d'instances différentes reprenant toutes les combinaisons possibles.

Preuve B.1 (démonstration du lemme 5.1)
1. Si le workflow est W= ($WA \subset \mathcal{AT}, Ax \subset \mathcal{A}xiomes$) une suite séquentielle d'activités WA= { A_i , 0<i<n+1 } sans comportement concurrentiel ni conditionnel alors il peut être représenté sous la forme d'une combinaison de patrons *séquence*. Ainsi selon la définition 4.10, les seules conditions décrivant les axiomes Ax du workflow sont :

$$\begin{gathered} A_1 < A_2 < A_3 < ..... < A_n \\ \wedge(\forall 1 < i < n; \texttt{OUT}_{max}(A_i) = \texttt{OUT}_{min}(A_i) = 1 \wedge \texttt{IN}_{max}(A_i) = \texttt{IN}_{min}(A_i) = 1) \\ \wedge(\texttt{OUT}_{max}(A_1) = \texttt{OUT}_{min}(A_1) = 1 \wedge \texttt{IN}_{max}(A_n) = \texttt{IN}_{min}(A_n) = 1) \end{gathered}$$

Par conséquent, seulement les points 2 et 3 de l'axiome 5.1 sont concernés.
La trace d'exécution $A_1A_2A_3.....A_n$ d'une instance d'exécution du workflow W satisfait amplement ces deux points.

2. Un comportement conditionnel entre n activités { A_i , 0<i<n+1 } et une activité B au niveau d'un point de «jointure» ou point de «diffusion» peuvent être décris par l'ensemble de patrons de «jointure» (*jointure simple*, *synchronisation* et *M-out-of-N*) ou de «diffusion» (*choix exclusif*, *diffusion parallèle*, *choix multiples*). On s'intéresse dans la suite de notre preuve aux patrons de «jointure», la preuve pour les patrons «diffusion» se fait symétriquement. Ainsi selon les définitions de ces patrons (décrite dans la section 4.3.5), les seules conditions possibles décrivant les axiomes Ax du workflow sont :

$$\forall 0 < i < n+1 A_i < B \wedge (\forall 1 < i < n; \texttt{OUT}_{max}(A_i) = \texttt{OUT}_{min}(A_i) = 1 \wedge \texttt{IN}_{min}(B) = 1)$$

Par conséquent, seulement les points 2 et 3 de l'axiome 5.1 sont concernés.

Les n traces d'exécutions sous la forme ...A_iB.... d'instances d'exécution du workflow satisfont amplement ces deux points.

3. Pour la preuve du troisième point, on s'intéresse en premier au comportement concurrentiel entre n flux de contrôle contenant chacun une activité. Ce comportement peut être décrit par un ensemble d'activités { A_i , 0<i<n+1 } concurrentes. Par conséquent, seulement le point 1 de l'axiome 5.1 est concerné.

L'ensemble des arrangements de taille n formés par l'ensemble d'activités { A_i , 0<i<n+1 } formant les $n! = \Pi_{i=1..n}(i)$ traces d'exécutions d'instances d'exécution du workflow satisfait amplement ce point.

Supposant maintenant que le flux k de contrôle contient plus qu'une activité et que { $A_{k,j}$, 0<j<m+1 }, alors il suffit de créer des arrangements de taille n formés par l'ensemble activités { $A_{k,j}$, A_i ; 0<i<k, k<i<n+1 } pour chaque activité $A_{k,j}$ formant ainsi les $\Pi_{0<i<k,k<i<n+1}(i)*(k-1+m)$ traces d'exécutions d'instances du workflow. En étendant notre supposition aux autres flux nous trouvons la formule du lemme $\Pi_{i=1..n}(j_i+i-1) = j_i*(j_2+1)*(j_3+2)*(j_4+3)*.....*(j_5+n-1)$.

B.2 Démonstration du théorème 5.1 sur la corrélation entre TDS et les dépendances d'activités

Énoncé du théorème 5.1

soit $wft = (WA \subset \mathcal{AT}, Ax \subset \mathcal{A}xiomes)$ un workflow transactionnel où les axiomes décrivent des patrons de workflow inclus dans les 7 patrons énoncés. $\forall a, b \in WA$ tel que $a \prec b \Rightarrow P(b/a) > 0$.

Preuve B.2 (Démonstration du théorème 5.1)
Selon la définition 4.6 :

$$\text{a} \prec \text{b} \stackrel{\text{def}}{=} depNrm(\text{a,b}) \stackrel{\text{def}}{=} dep(a_1.\texttt{terminer()}, a_2.\texttt{activer()})$$

et comme nous ne traitons que des instances d'exécution sans échecs, nous pouvons conclure en se basant sur les prédicats du domaine de définition d'une activité transactionnelle (voir tableau 4.2) que chaque activation d'une activité se termine toujours avec succès, ainsi l'équivalence précédente peut être réécrite comme suit :

$$\text{a} \prec \text{b} \stackrel{\text{def}}{=} depNrm(\text{a,b}) \stackrel{\text{def}}{=} dep(a_1.\texttt{terminer()}, a_2.\texttt{terminer()})$$

Les axiomes du workflow transactionnel décrivent un des 7 des patrons de workflow énoncés. Si ces axiomes proviennent du patron *séquence*, nous pouvons ajouter les axiomes suivants :

$$\texttt{OUT}_{max}(a) = \texttt{OUT}_{min}(a) = 1 \wedge \texttt{IN}_{max}(b) = \texttt{IN}_{min}(b) = 1)$$

De même s'ils viennent de patrons de «diffusion» :

$$\texttt{IN}_{max}(b) = \texttt{IN}_{min}(b) = 1)$$

ou de patrons de «jointure» :

$$\texttt{OUT}_{max}(a) = \texttt{OUT}_{min}(a) = 1)$$

Ainsi, en appliquant les conclusions de l'axiome 5.1, nous pouvons déduire que :

$$\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < n, |t_i = a, t_{i+1} = b$$

Et par la construction de la table statistique de dépendances et en se basant sur cette trace d'exécution, on peut conclure que $P(b/a) > 0$.

B.3 Démonstration du lemme 5.2 sur la corrélation entre TDS et les dépendances inter-activités

Énoncé du théorème 5.1
soit $wft = (WA \subset \mathcal{AT}, Ax \subset \mathit{Axiomes})$ un workflow transactionnel sans boucle de taille égale à 1 où les axiomes décrivent des patrons de workflow inclus dans les 7 patrons énoncés. $\forall a, b \in WA$ tel que $a \prec b \Leftrightarrow P(b/a) > 0 \wedge P(a/b) = 0$

Preuve B.3 (Démonstration du lemme 5.2)
Preuve du premier sens d'implication «⇒» (démonstration par l'absurde)

$$a \prec b \Rightarrow P(b/a) > 0$$

supposons maintenant que $P(a/b) > 0$ ceci implique :

$$\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < n, |t_i = b, t_{i+1} = a$$

et comme nous avons déjà $P(b/a) > 0$ ce qui implique :

$$\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < n, |t_i = a, t_{i+1} = b$$

Or ceci est absurde car ce cas ne se présenta que dans une boucle de taille égale à deux ou dans le cas où nous avons des activités concurrentes selon le premier point de l'axiome de traces d'exécutions complètes (axiome 5.1). Ainsi nous avons :

$$a \prec b \Leftarrow P(b/a) > 0 \wedge P(a/b) = 0$$

Preuve du deuxième sens d'implication «⇐» (démonstration par l'absurde)
Nous avons $P(b/a) > 0 \wedge P(a/b) = 0$ par construction de la table statistique de dépendances nous

$$\exists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < n, |t_i = a, t_{i+1} = b$$
$$wedge \nexists \sigma_1 = t_1 t_2 t_3 ....t_{n-1} \in L \wedge \exists 0 < i < n, |t_i = a, t_{i+1} = b \ \textbf{(1)}$$

Les axiomes du workflow transactionnel décrivent un des 7 des patrons de workflow énoncés. Si les deux activités a et b sont en indépendance causale, deux sous cas se présentent :

1. les deux activités a et b appartiennent à deux patrons disjoints. Ceci est absurde d'après la trace d'exécution **(1)** qui montrent que les deux activités se suivent.
2. les deux activités a et b sont concurrentes. Ceci est absurde d'après la trace d'exécution **(1)** et en se basant sur l'axiome de traces d'exécutions complètes (axiome 5.1)).

Ainsi les deux activités a et b dépendent l'une de l'autre. Le cas $b \prec a$ est trivialement absurde d'après le théorème 5.1. En conclusion nous avons :

$$a \prec b \Leftarrow P(b/a) > 0 \wedge P(a/b) = 0$$

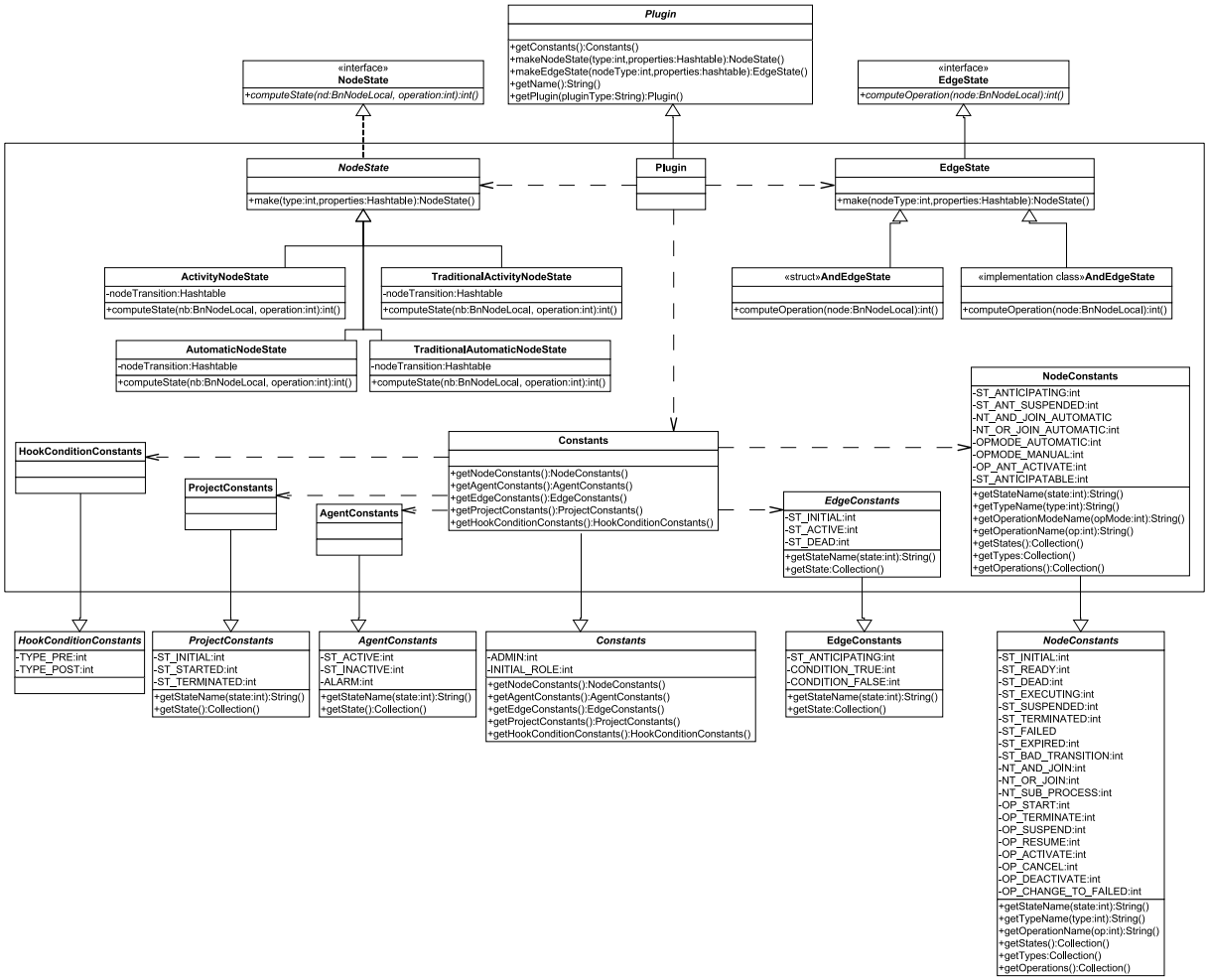
Annexe C

Diagrammes UML

Cette annexe contient une présentation plus détaillée par des diagrammes d'activités UML de l'architecture logicielle des différentes applications développées.

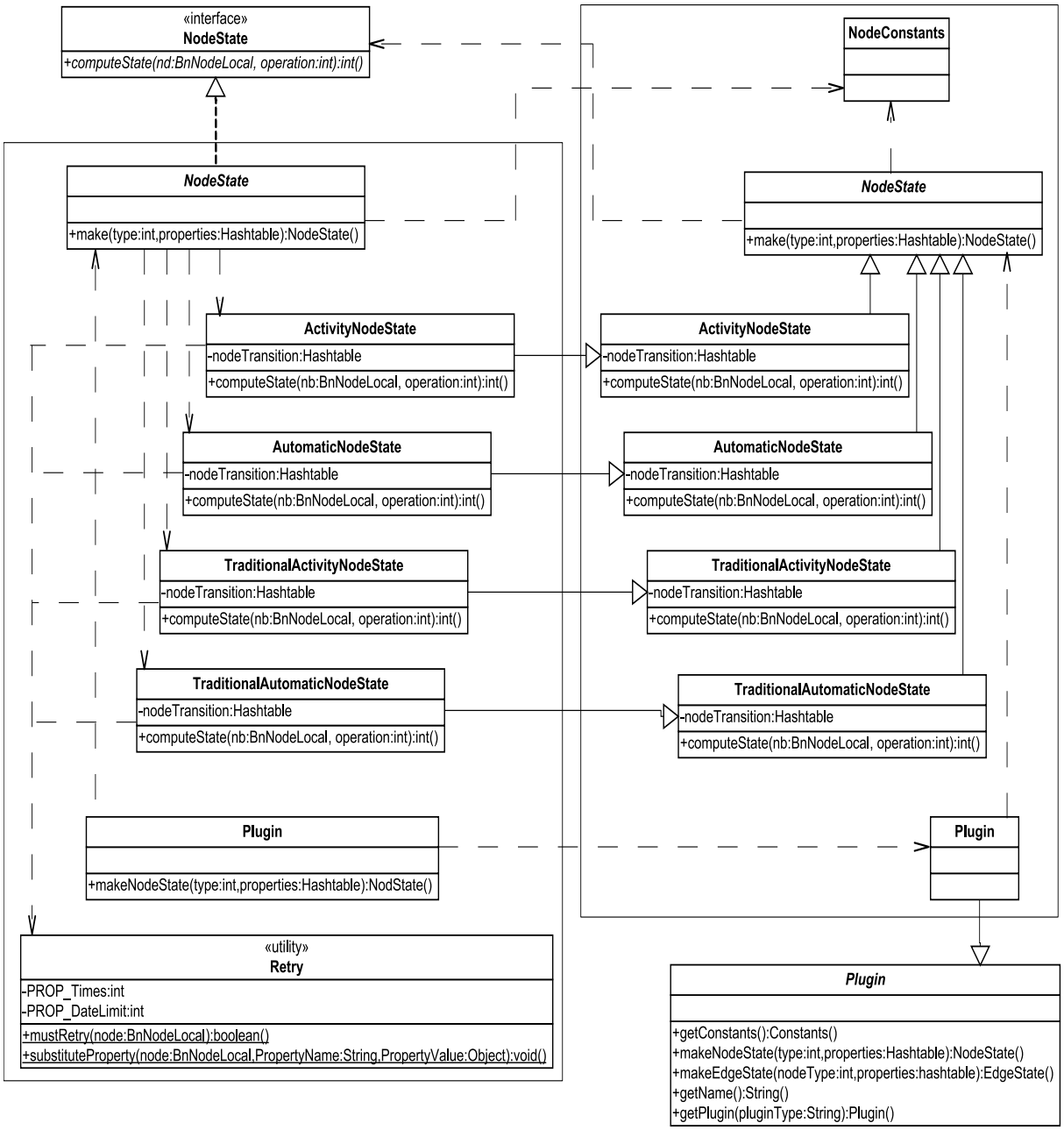
C.1 Plug-in transactionnel dans `Bonita`

Figure C.5 Diagramme de classes de plug-in.



Le diagramme UML de la figure C.5 détaille l'architecture d'un plug-in dans `Bonita` présenté sous sa forme de base dans la figure 6.2. Notons que cette architecture intègre les éléments de base décrits dans la section 6.2.2.3.

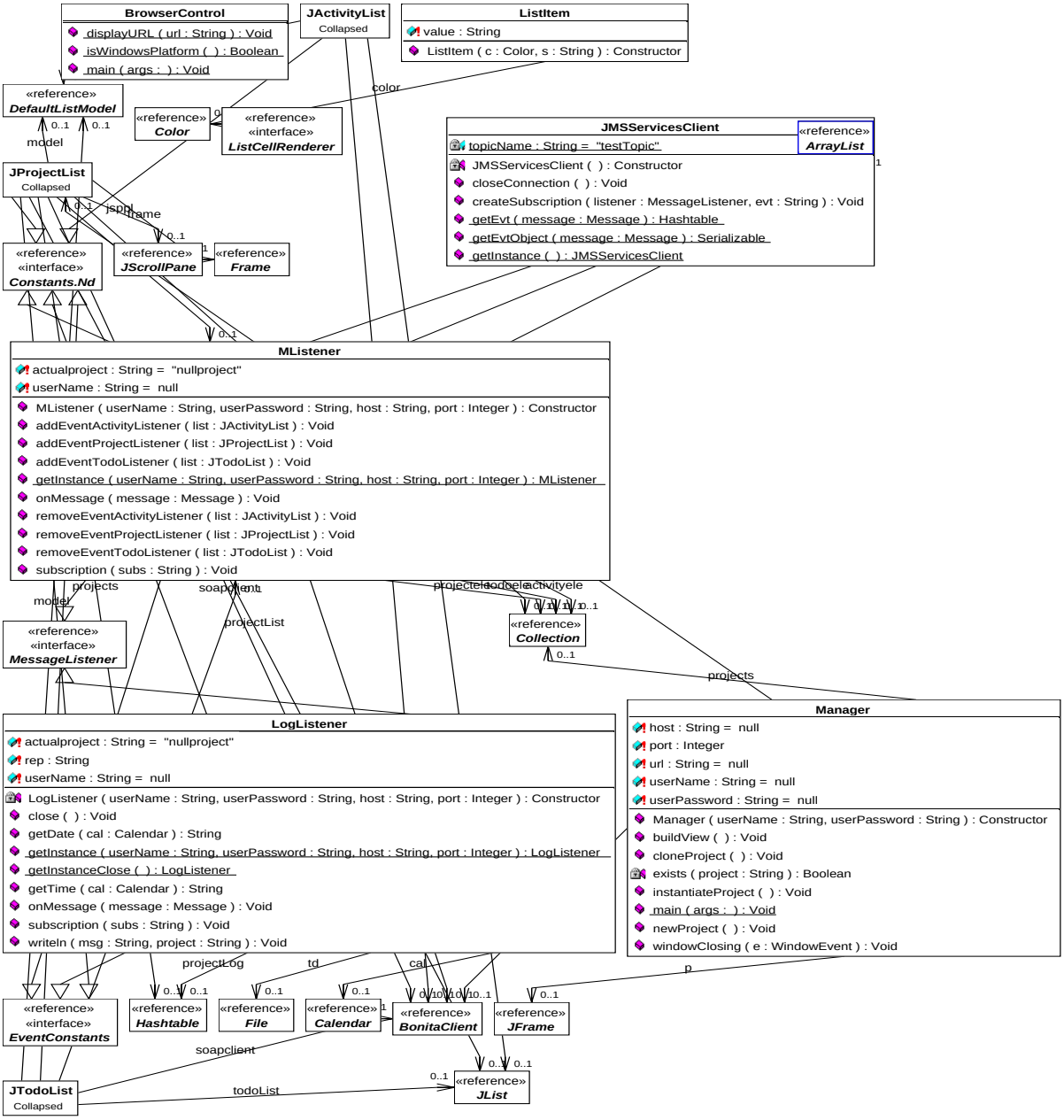
Figure C.6 Le plug-in rejouable implémenté comme extension du plug-in de base.



La figure C.6 présente le diagramme de classes du plug-in «rejouable». Elle illustre comment ses classes (encadrées par le rectangle à gauche) étendent les classes du plug-in de base (encadrées par le rectangle à droite).

C.2 Collecte de traces d'exécutions dans Bonita

Figure C.7 Diagramme de classes de l'API de collecte de de traces d'exécution dans Bonita

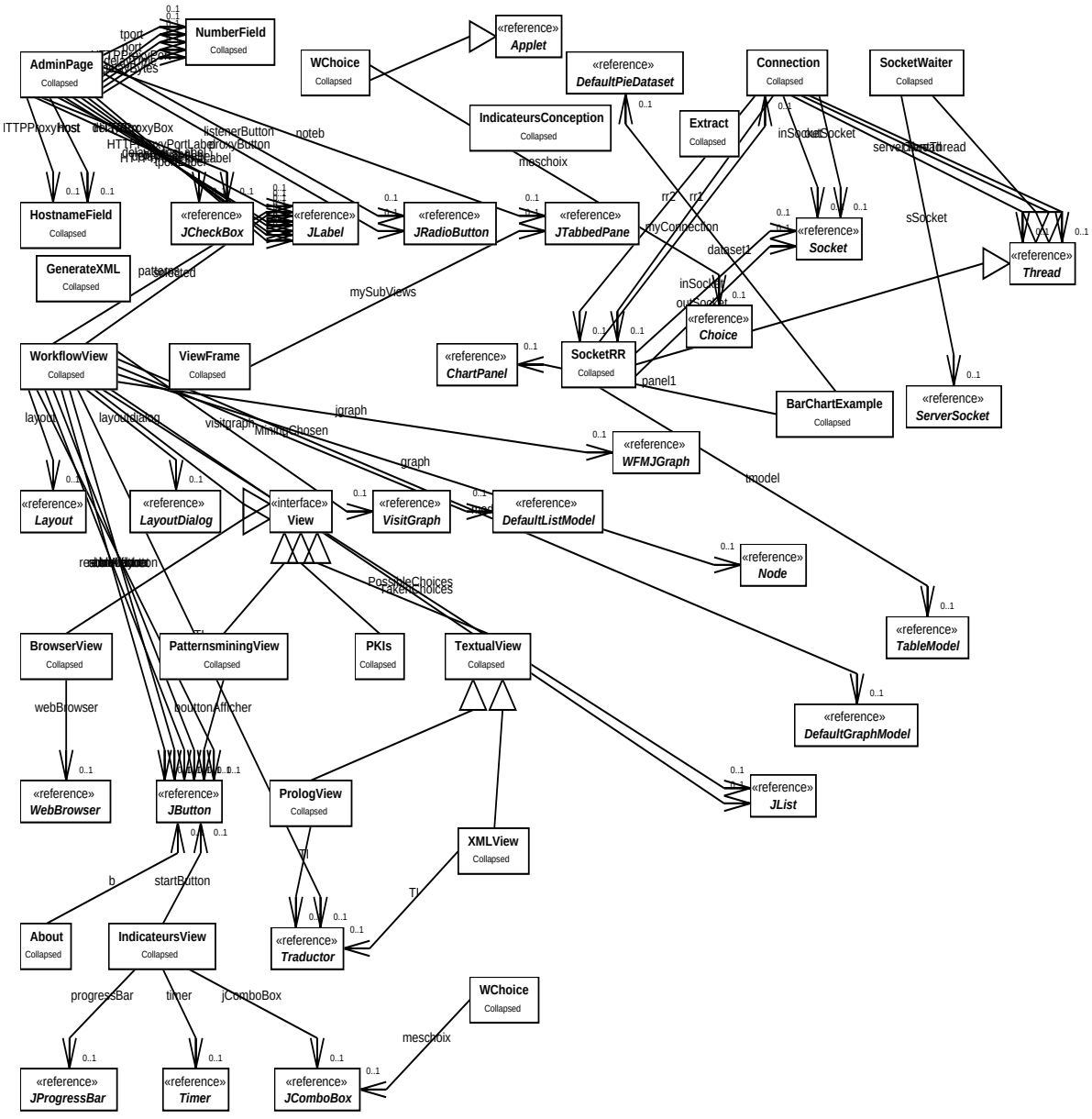


La figure C.7 présente le diagramme UML de l'API de collecte de traces d'exécution dans Bonita. Notons que cette API se base sur le service JMS de Bonita (*c.f.* la classe «JMSServicesCLENT») pour la collecte les messages échangés dans le moteur d'exécution.

C.3 Environnement de découverte de workflow

C.3.1 `Workflowminer`

Figure C.8 Diagramme de classes de l'architecture générale de `Workflowminer`



La figure C.8 présente le diagramme de classes en UML de l'architecture générale de notre outil de découverte de workflow `Workflowminer`. Dans ce diagramme, nous pouvons distinguer les différents vues de notre outil représentées par les classes «PatternsminingView», «PKIs», «PrologView» et «XMLView».

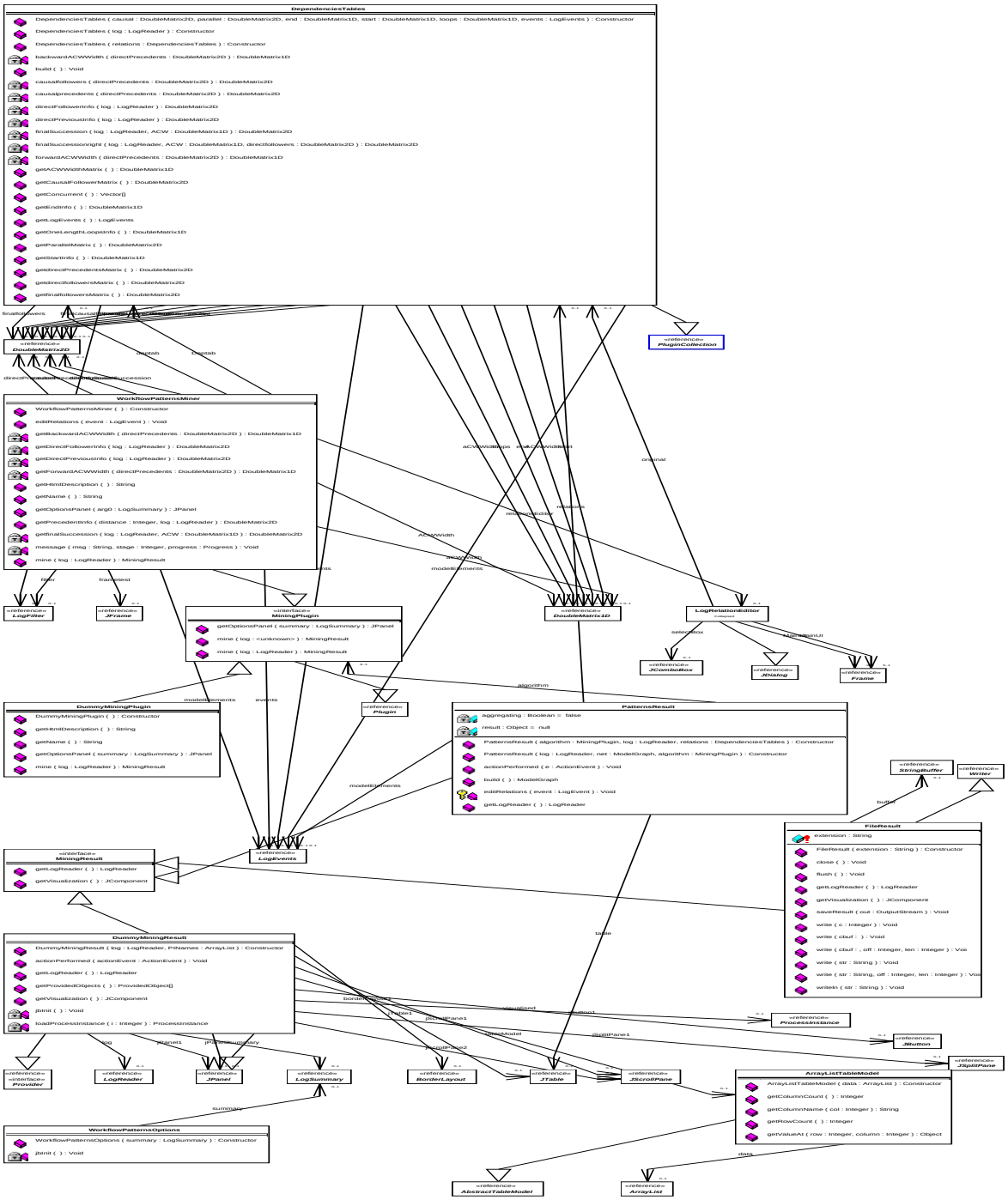
Figure C.9 Diagramme de classes de l'interface graphique de la vue «Patterns VIEW»

La figure C.9 présente le diagramme de classes en UML de l'interface graphique de la vue «Patterns VIEW». L'utilisation du patron de conception Visiteur pour la conception de cette vue nous a permit de séparer l'algorithme de visualisation de l'objet graphique du contenu de l'objet représenté.

C.3.2 Plug-in «`Workflow patterns miner`»

La figure C.10 présente le diagramme de classes en UML de l'architecture générale de notre plug-in de découverte de workflow «`Workflow patterns miner`». Notons que notre plug-in hérite de l'interface «Mining Plugin» de `ProM`.

Figure C.10 Diagramme de classes du plug-in «`Workflow patterns miner`»



Annexe D

Tests de validation

Cette annexe contient un ensemble exhaustif d'exemples de workflows que nous avons utilisés pour appliquer et tester les outils de simulation de traces d'exécutions et de découverte de workflow.

D.1 Traces d'exécutions simulées par les outils CPN

Dans la suite, nous montrons un extrait des traces d'exécutions du workflow représenté dans le tableau D.2. Ces traces d'exécutions ont été obtenues suite à une simulation aléatoire du workflow par les outils CPN. Elles représentent l'exécution des deux instances numéros 71 et 72. Notons que structure XML suit le format proposée dans la figure 5.4.

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- <WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://tmitwww.tm.tue.nl/research/processmining/WorkflowLog.xsd"
description="Log extracted from CPN Tools">
  <Source program="CPN Tools" />
- <Process id="CpnToolsLog" description="Log file created in CPN Tools"> -
<ProcessInstance id="71" description=""> - <AuditTrailEntry>
  <WorkflowModelElement>Start</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>A</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>C</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>B</WorkflowModelElement>
  <EventType>complete</EventType>
```

```
  <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>D</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>E</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:02:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>G</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:02:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>End</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:02:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
  </ProcessInstance>
- <ProcessInstance id="72" description=""> - <AuditTrailEntry>
  <WorkflowModelElement>Start</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>A</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>B</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>D</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>C</WorkflowModelElement>
```

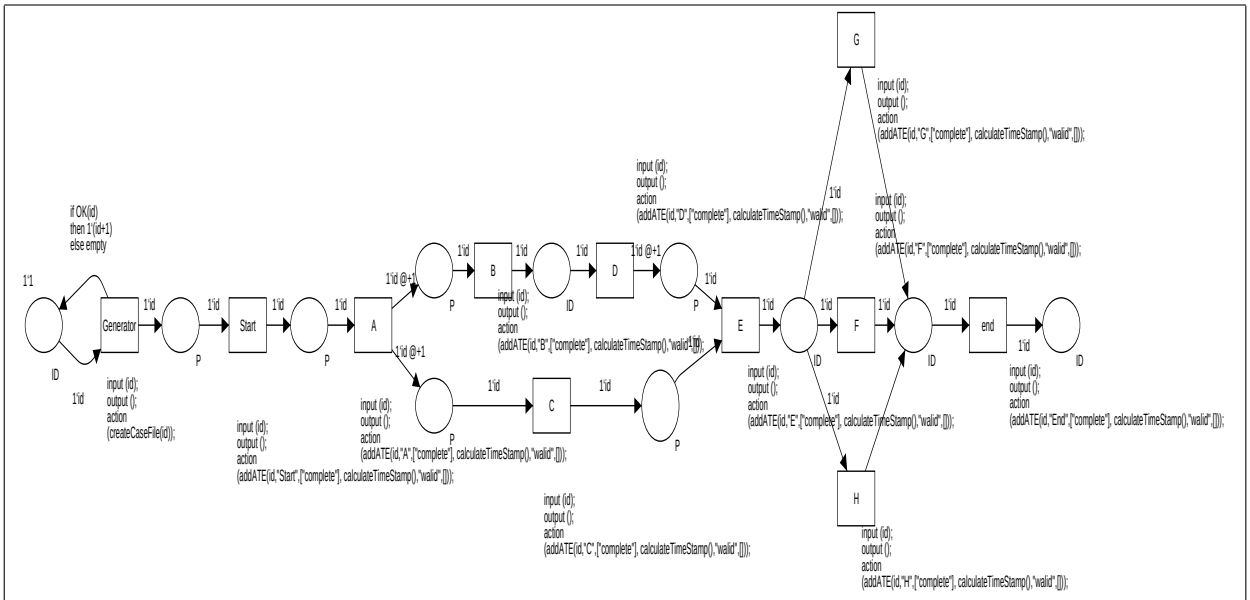
```
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>E</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:02:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>G</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:02:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>End</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:02:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
  </ProcessInstance>
- <ProcessInstance id="73" description=""> - <AuditTrailEntry>
  <WorkflowModelElement>Start</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>A</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>0000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>walid</Originator>
  </AuditTrailEntry>
- <AuditTrailEntry>
  <WorkflowModelElement>C</WorkflowModelElement>
  <EventType>complete</EventType>
......
```

D.2 Examples de workflows découverts par «Workflow patterns miner»

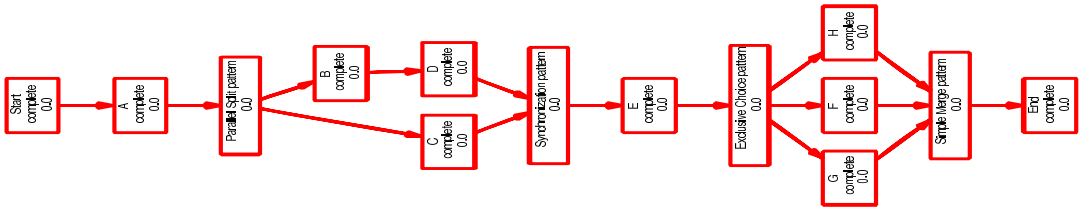
Dans cette section, nous présentons les résultats des expériences de découverte exécutées dans «Workflow patterns miner» sur des traces d'exécutions de workflows exemples simulés par les outils CPN. À savoir, nous avons utilisé cinq workflows CPN divers avec un niveau complexité variée, qui couvrent différents types de point de «jointure» et de «diffusion», des flux parallèles, et des cycles. Notre but est de redécouvrir ces cinq modèles de workflows à partir de leurs traces d'exécutions. Notons que les workflows simulés sont représentés à l'aide des réseaux de pétri et que les workflows découvert sont représentés à l'aide des patrons de workflow et que les deux modèles ont un comportement équivalent.

D.2.1 Points de «jointure» et de «diffusion»

Le tableau D.1 rapporte un workflow qu'on a traité dans notre papier [GG06]. Ce workflow comprend 2 points de «diffusion» et 2 points de «jointure» différents représentant 4 patrons de workflow : *diffusion parallèle*, *choix exclusif*, *synchronisation*, *jointure simple*.



Workflow simulé par CPN

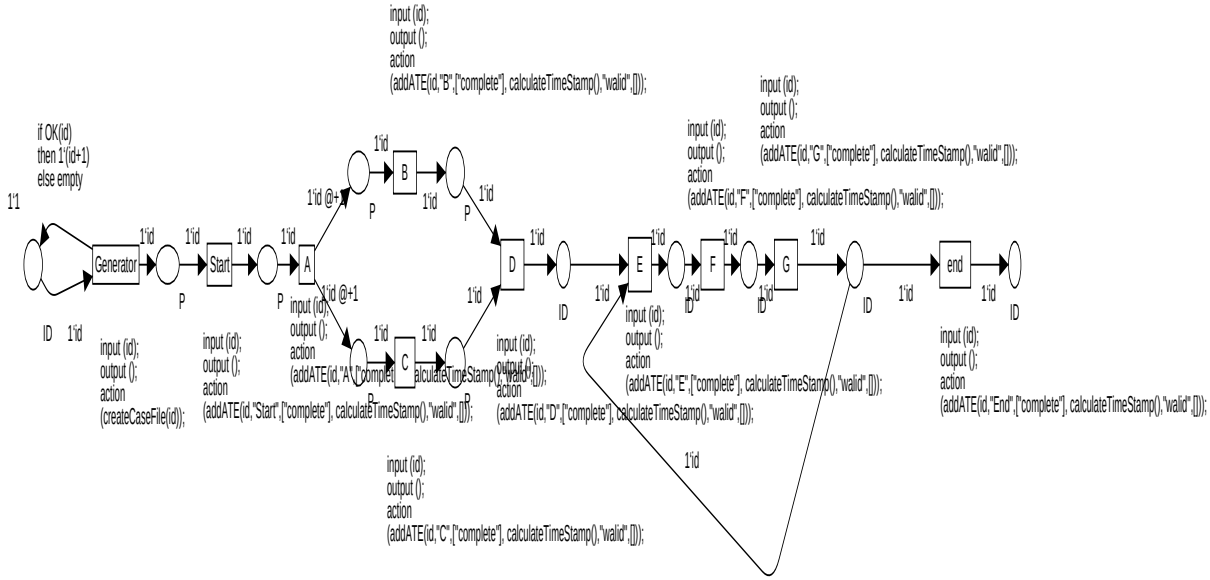


Workflow découvert par «`Workflow patterns miner`»

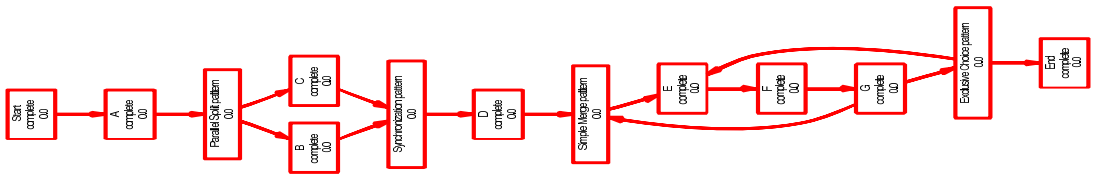
TAB. D.1 – Workflow à 4 points de «jointure» et de «diffusion» différents

D.2.2 Boucle ordinaire

Le tableau D.2 rapporte un workflow contenant une boucle de trois activités (c.à.d. les activités E, F, G). La représentation de cette boucle dans le workflow découvert se fera à l'aide des patrons de workflow : *choix exclusif* et *jointure simple*.



Workflow simulé par CPN

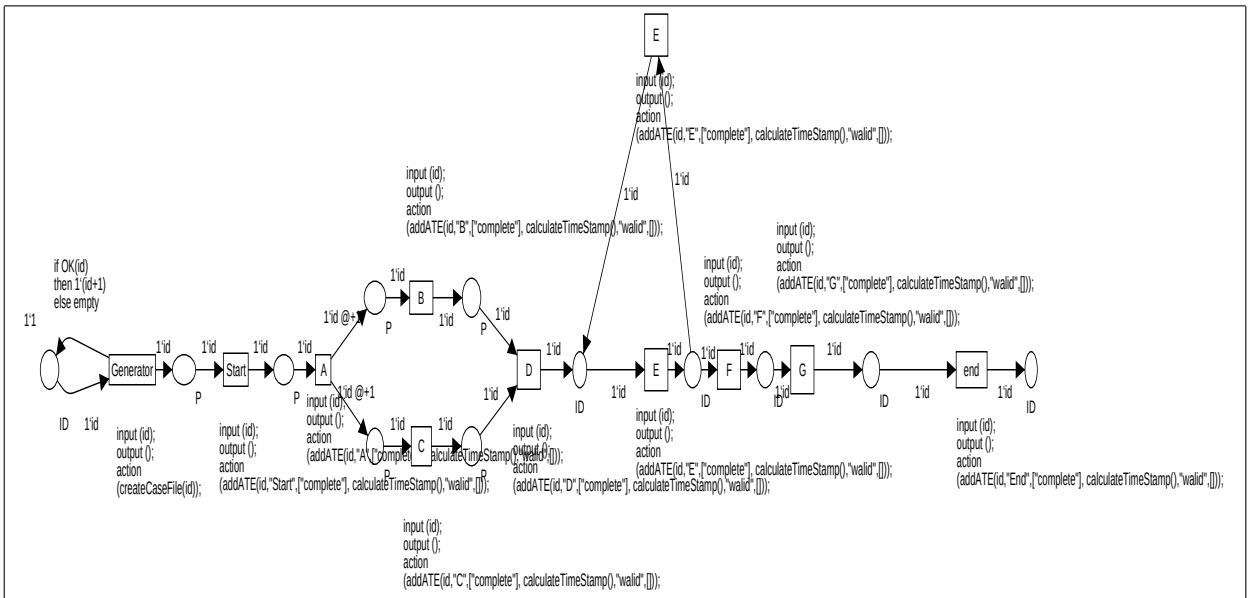


Workflow découvert par «Workflow patterns miner»

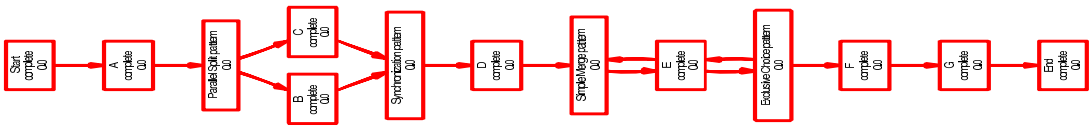
TAB. D.2 – Workflow contenant une boucle «ordinaire»

D.2.3 Boucle courte

Le tableau D.3 rapporte un workflow contenant une boucle courte d'une activité (c.à.d. l'activité E). De même que dans le tableau D.2, la représentation de cette boucle dans le workflow découvert se fera à l'aide des patrons de workflow : *choix exclusif* et *jointure simple*. Notons que notre algorithme de découverte de patrons workflow traite tout type de boucle sauf les boucles de taille égale à 2 qui nécessite un pré-traitement spécifique ; Cela est du caractère spéciale de ce comportement qui peut être confondu avec un comportement concurrentiel dans les traces d'exécutions.



Workflow simulé par CPN

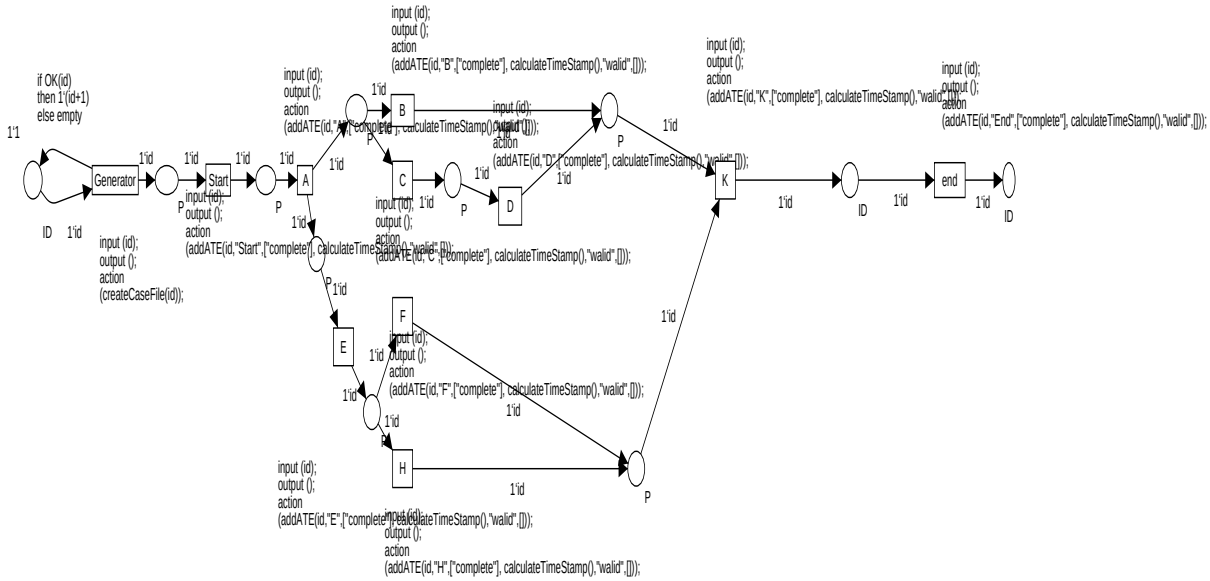


Workflow découvert par «`Workflow patterns miner`»

TAB. D.3 – Workflow contenant une boucle «courte» de taille 1

D.2.4 Choix non-libre

Le tableau D.4 rapporte un patron de workflow particulier qui est le patron *M-out-of-N-Join*. Ce patron décrit le comportement spécial de choix non-libre qui est à la fois un mélange de synchronisation et de choix dans un seul opérateur. Le patron *M-out-of-N-Join* représente une implémentation possible de ce comportement [vdABtHK00].



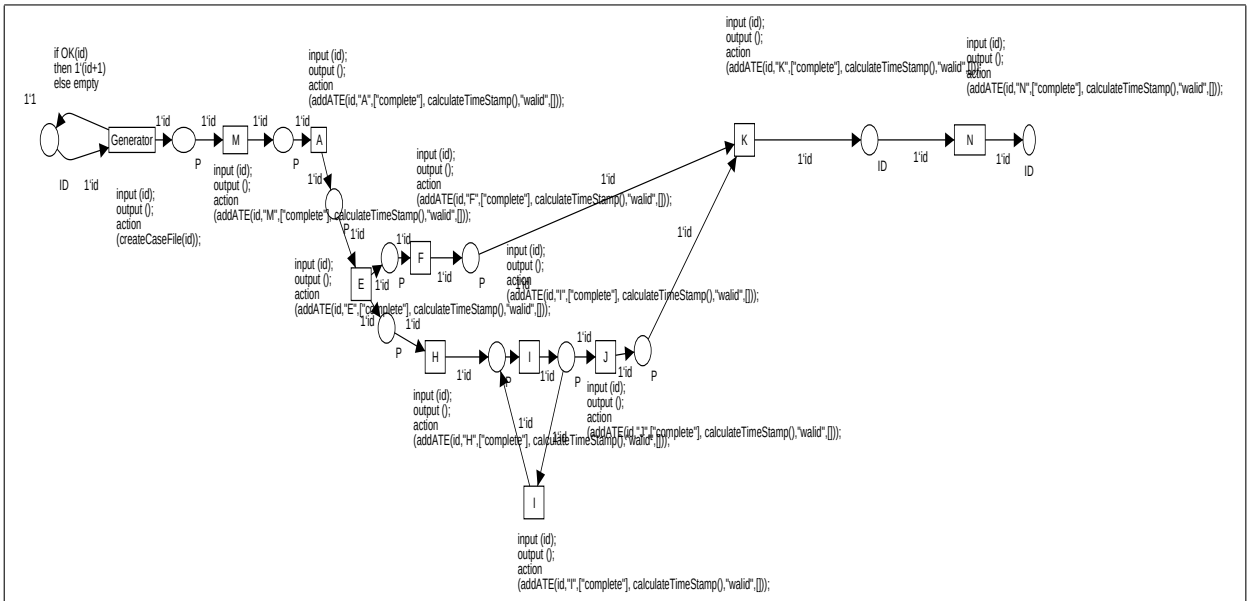
Workflow simulé par CPN

Workflow découvert par «`Workflow patterns miner`»

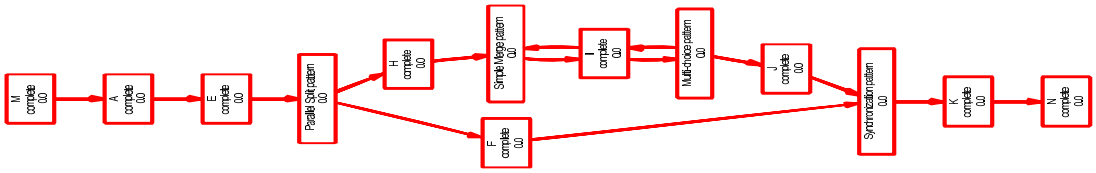
TAB. D.4 – Workflow contenant le patron de workflow *M-out-of-N-Join*

D.2.5 Boucle et concurrence

Le tableau D.5 rapporte un workflow contenant une boucle qui est mise en parallèle (c.à.d. l'activité I) avec un flux de contrôle concurrent (c.à.d. le flux contenant l'activité F). L'originalité de workflow vient de la complexité ou la difficulté de séparer le comportement de boucle et le comportement concurrentiel d'une seule activité.



Workflow simulé par CPN



Workflow découvert par «`Workflow patterns miner`»

TAB. D.5 – Workflow contenant une boucle avec un comportement concurrentiel

Bibliographie

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In Stanley Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 574–581. IEEE Computer Society, 1996.

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert - Special Issue on Cooperative Information Systems*, 12(5), 1997.

[ABJ+04] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler : An extensible system for design and execution of scientific workflows, 2004.

[AGL98] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *EDBT*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer, 1998.

[AM97] G. Alonso and C. Mohan. Workflow Management Systems : The Next Generation of Distributed Processing Tools. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, pages 35–62. Kluwer Academic Publishers, 1997.

[ARS92] N. Ansari, L. Rusinkiewicz, and A. Sheth. Using flexible transaction to support multi-system telecommunication applications. In *Proc. of the 18th VLDB*, pages 65–76, Vancouver, Canada, 1992.

[AS83] D. Angluin and C. H. Smith. Inductive inference : Theory and methods. *ACM Comput. Surv.*, 15(3) :237–269, 1983.

[Bai03] K. Baina. *Un Modèle Orienté Services Procédés pour l'Interconnexion et la Coopération des Procédés d'Entreprises.* Phd thesis, Université Henri Poincaré - Nancy 1, LORIA, May 16, 2003.

[BCWH+03] B. Benatallah, P. Chrzastowski-Wachtel, R. Hamadi, M. O'Dell, and A. Susanto. Hiword : A petri net-based hierarchical workflow designer. In *ACSD*, pages 235–236. IEEE Computer Society, 2003.

[BDG+94] A. Biliris, S. Dar, N. Gehani, H. V. Jagadish, and K. Ramamritham. Asset : a system for supporting extended transactions. *SIGMOD Rec.*, 23(2) :44–54, 1994.

[BGG06] S. Bhiri, W. Gaaloul, and C. Godart. Discovering and improving recovery mechanisms of composite web services. In *The 2006 IEEE International Conference on Web Services (ICWS 2006)*, IEEE Press, Chicago, USA, September 18-22, 2006. IEEE Computer Society.

[BGKM06] K. Baïna, W. Gaaloul, R. El Khattabi, and A. Mouhou. A new workflow patterns and performance analysis tool. In *CAiSE'06 Forum and Tools Demonstrations Proccedings*, LNCS, Luxembourg, Luxembouorg, June 9 2006. Springer-Verlag.

[BGP04a] S. Bhiri, C. Godart, and O. Perrin. A transaction-oriented framework for composing transactional web services. In *IEEE SCC*, pages 654–663. IEEE Computer Society, 2004.

[BGP04b] S. Bhiri, C. Godart, and O. Perrin. A transactional-oriented framework for composing transactional web services. In *IEEE International Conference on Services Computing (SCC 2004)*. IEEE Computer Society, 2004.

[BPG05] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 138–147. ACM, 2005.

[BPG06] S. Bhiri, O. Perrin, and C. Godart. Extending workflow patterns with transactional dependencies to define reliable composite web services. In *AICT/ICIW*, page 145. IEEE Computer Society, 2006.

[BvdA01] T. Basten and W. M. P. van der Aalst. Inheritance of behavior. *J. Log. Algebr. Program.*, 47(2) :47–145, 2001.

[BZB97] O. A. Bukhres, P. Zhang, and B. Benatallah. Modeling of dynamic internet transactional workflows. In *ADBIS*, pages 379–389. Nevsky Dialect, 1997.

[CD01] J. E. Cook and Z. Du. Discovering models of behavior for concurrent systems. In *Proceedings of the 2001 International Conference on Software Mainenance*, Florence, Italy, November 2001.

[Coa96] WorkFlow Management Coalition. Terminology and glossary. technical report wfms-tc-1011. Technical report, Workflow Management Coalition Brussels - Belgium, 1996.

[Coa03] DAML-S Coalition. Daml-s 0.9 draft release. http ://www.daml.org/services/daml-s/0.9/, 2003.

[Cor02] Microsoft Corporation. Microsoft biztalk server 2002 enterprise edition. http ://www.microsoft.com, 2002.

[CR94] P. K. Chrysanthis and K. Ramamritham. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, 19(3) :450–491, 1994.

[CW94] J. E. Cook and A. L. Wolf. Toward metrics for process validation. In *Proceedings of the Third International Conference on the Software Process*, pages 33–44, Reston, Virginia, 10–11 1994. IEEE.

[CW95] J. E. Cook and A. L. Wolf. Automating process discovery through event-data analysis. In *Proceedings of the 17th international conference on Software engineering*, pages 73–82. ACM Press, 1995.

[CW98a] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3) :215–249, 1998.

[CW98b] J. E. Cook and A. L. Wolf. Event-based detection of concurrency. In *Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 35–45. ACM Press, 1998.

[CW98c] J. E. Cook and A. L. Wolf. Event-based detection of concurrency. In *6th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM Press, 1998.

[CW99] J. E. Cook and A. L. Wolf. Software process validation : quantitatively measuring the correspondence of a process to a model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(2) :147–176, 1999.

[CWBH+03] P. Chrzastowski-Wachtel, B. Benatallah, R. Hamadi, M. O'Dell, and A. Susanto. A top-down petri net-based approach for dynamic workflow modeling. In *Business Process Management*, pages 336–353, 2003.

[DDGJ01] W. Derks, J. Dehnert, P. Grefen, and W. Jonker. Customized atomicity specification for transactional workflow. In *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'01)*, pages 140–147. IEEE Computer Society, 2001.

[DDS97] W. Du, J. Davis, and M. C. Shan. Flexible specification of workflow compensation scopes. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge*, pages 309–316. ACM Press, 1997.

[Deh01] J. Dehnert. Four systematic steps towards sound business process models. In *Proc. of 2nd Int. Coll. on Petri Net Technologies for Modelling Communication Based Systems / Weber, Ehrig, Reisig (Eds.)*, pages 55–64. DFG Research Group "Petri Net Technology", September 2001.

[DFDB05] H. Duarte, M.-C. Fauvet, M. Dumas, and B. Benatallah. Vers un modèle de composition de services web avec propriétés transactionnelles. *Ingénierie des Systèmes d'Information*, 10(3) :9–28, 2005.

[DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 204–214. ACM Press, 1990.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A transactional model for long-running activities. In Guy M. Lohman, Amílcar Sernadas, and Rafael Camps, editors, *17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings*, pages 113–122. Morgan Kaufmann, 1991.

[DK00] C. Dellarocas and M. Klein. A knowledge-based approach for designing robust business processes. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 50–65, London, UK, 2000. Springer-Verlag.

[DKRR98] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *PODS '98 : Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 25–33, New York, NY, USA, 1998. ACM Press.

[dMvDvdAW04] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining for ubiquitous mobile systems : An overview and a concrete algorithm. In Luciano Baresi, Schahram Dustdar, Harald Gall, and Maristella Matera, editors, *UMICS*, volume 3272 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2004.

[dMWvdA05] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic process mining : A basic approach and its challenges. In *Business Process Management Workshops*, pages 203–215, 2005.

[DS90] T.H. Davenport and J.E. Short. The new industrial engineering : Information technology and business process redesign. *Sloan Management Review*, 31(4) :4–10, 1990.

[(Ed92] A. Elmagarmid (Ed.). *Transaction Models for Advanced Database Applications*. Morgan-Kaufmann, 1992.

[EKR95] C. A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *COOCS*, pages 10–21. ACM, 1995.

[Ell79] C.A. Ellis. Information control nets : A mathematical model of office information flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.

[Ell99] C. A. Ellis. *Computer Supported Cooperative Work*, chapter Workflow Technology. John Wiley and Sons, 1999.

[ELLR90] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multidatabase transaction model for interbase. In *Proceedings of the sixteenth international conference on Very large databases*, pages 507–518. Morgan Kaufmann Publishers Inc., 1990.

[EOG02] J. Eder, G. E. Olivotto, and W. Gruber. A data warehouse for workflow logs. In *Proceedings of the First International Conference on Engineering and Deployment of Cooperative Information Systems*, pages 1–15. Springer-Verlag, 2002.

[FDDB05] M.-C. Fauvet, H. Duarte, M. Dumas, and B. Benatallah. Handling transactional properties in web service composition. In Anne H. H. Ngu, Masaru Kitsuregawa, Erich J. Neuhold, Jen-Yao Chung, and Quan Z. Sheng, editors, *WISE*, volume 3806 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2005.

[FG94] R. Focardi and R. Gorrieri. A classification of security properties for process algebra, 1994.

[Fis00] L. Fischer, editor. *The Workflow Handbook 2001*. Published in association with the Workflow Management Coalition (WfMC), October 2000.

[GBBG04] W. Gaaloul, K. Baïna, K. Benali, and C. Godart. A pattern for interconnecting distributed components. In *ICEIS2004, Proceedings of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, April 14-17, 2004*, pages 430–434, 2004.

[GBG04a] W. Gaaloul, S. Bhiri, and C. Godart. Discovering workflow patterns from timed logs. In *EMISA 2004, Informationssysteme im E-Business und E-Government, Beiträge des Workshops der GI-Fachgruppe EMISA*, LNI, pages 84–94, Luxemburg, October 6-8, 2004. GI.

[GBG04b] W. Gaaloul, S. Bhiri, and C. Godart. Discovering workflow transactional behaviour event-based log. In *12th International Conference on Cooperative Information Systems (CoopIS'04)*, LNCS, Larnaca, Cyprus, October 25-29, 2004. Springer-Verlag.

[GBG05] W. Gaaloul, K. Baïna, and C. Godart. Towards mining structural workflow patterns. In K. V. Andersen, J. K. Debenham, and R. Wagner, editors, *DEXA*, volume 3588 of *LNCS*, pages 24–33. Springer, 2005.

[GBG06] W. Gaaloul, K. Baïna, and C. Godart. A bottom-up workflow mining approach for workflow applications analysis. In *The 2nd International Workshop on Data Engineering Issues in E-Commerce and Services*, LNCS, San Francisco, California, USA, June 26 2006. Springer-Verlag.

[GC02] N. Gioldasis and S. Christodoulakis. Utml : Unified transaction modeling language. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering*, pages 115–126. IEEE Computer Society, 2002.

[GCC+04] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M-C. Shan. Business process intelligence. *Comput. Ind.*, 53(3) :321–343, 2004.

[GdV98] P. W. P. J. Grefen and R. R. de Vries. A reference architecture for workflow management systems. *Data Knowl. Eng.*, 27(1) :31–57, 1998.

[GG05] W. Gaaloul and C. Godart. Mining workflow recovery from event based logs. In *Business Process Management*, pages 169–185, 2005.

[GG06] W. Gaaloul and C. Godart. A workflow mining tool based on logs statistical analysis. In Frank Maurer and Günther Ruhe, editors, *SEKE*, pages 37–44, 2006.

[GGM+04] G. Greco, A. Guzzo, G. Manco, L. Pontieri, and D. Saccà. Mining constrained graphs : The case of workflow systems. In Jean-François Boulicaut, Luc De Raedt, and Heikki Mannila, editors, *Constraint-Based Mining and Inductive Databases*, volume 3848 of *Lecture Notes in Computer Science*, pages 155–171. Springer, 2004.

[GGMS05] G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining and reasoning on workflows. *IEEE Trans. Knowl. Data Eng.*, 17(4) :519–534, 2005.

[GGP05] G. Greco, A. Guzzo, and L. Pontieri. Mining hierarchies of models : From abstract views to concrete specifications. In *Business Process Management*, pages 32–47, 2005.

[GGPS04] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Mining expressive process models by clustering workflow traces. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 52–62. Springer, 2004.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlisside. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[GHKM94] D. Georgakopoulos, M. F. Hornick, P. Krychniak, and F. Manola. Specification and management of extended transactions in a programmable transaction environment. In *Proceedings of the Tenth International Conference on Data Engineering, February 14-18, 1994, Houston, Texas, USA*, pages 462–473. IEEE Computer Society, 1994.

[GMGK+91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Modeling long-running activities as nested sagas. *IEEE Data Eng. Bull.*, 14(1) :14–18, 1991.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 249–259. ACM Press, 1987.

[GP03] M. Golani and S. S. Pinter. Generating a process model from a process audit log. In *Business Process Management*, pages 136–151, 2003.

[GPS99] P. Grefen, B. Pernici, and G. Sanchez, editors. *Database Support for Workflow Management : The Wide Project*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[Gra] GraphViz : A Graph Vizualization Package. AT&T Research.

[GVA01] P. Grefen, J. Vonk, and P. Apers. Global transaction support for workflow management systems : from formal specification to practical implementation. *The VLDB Journal*, 10(4) :316–333, 2001.

[GVBA97] P. W. P. J. Grefen, J. Vonk, E. Boertjes, and P. M. G. Apers. Two-layer transaction management for workflow management applications. In *DEXA '97 : Proceedings of the 8th International Conference on Database and Expert Systems Applications*, pages 430–439, London, UK, 1997. Springer-Verlag.

[Ham90] M. Hammer. Reengineering work : Don't automate, obliterate. *Harvard Business Review*, pages 70–91, 1990.

[Han94] G. Hansen. *Automating Business Process Re-engineering : Using the Power of Visual Simulation Strategies to Improve Performance and Profit*. Prentice Hall, New York, 1994.

[Har03] E. R. Harold. *Processing XML with Java : a guide to SAX, DOM, JDOM, JAXP, and TrAX*. 2003.

[HB04] R. Hamadi and B. Benatallah. Recovery nets : Towards self-adaptive workflow systems. In Xiaofang Zhou, Stanley Y. W. Su, Mike P. Papazoglou, Maria E. Orlowska, and Keith G. Jeffery, editors, *WISE*, volume 3306 of *Lecture Notes in Computer Science*, pages 439–453. Springer, 2004.

[HC93] M. Hammer and J. Champy. Reengineering the corporation : A manifesto for business revolution. *Business Horizons*, 36(5) :90–91, 1993. available at http ://ideas.repec.org/a/eee/bushor/v36y1993i5p90-91.html.

[Her00a] J. Herbst. Dealing with concurrency in workflow induction. In Machine Learning : ECML 2000, editor, *11th European Conference on Machine Learning, Barcelona, Catalonia, Spain*, number Vol 1810 in Lecture Notes in Artificial Intelligence, pages 183–194. Springer-Verlag, May 2000.

[Her00b] J. Herbst. An inductive approach to the acquisition and adaptation of workflow models. In Machine Learning : ECML 2000, editor, *11th European Conference on Machine Learning, Barcelona, Catalonia, Spain*, number Vol 1810 in Lecture Notes in Artificial Intelligence, pages 183–194. Springer-Verlag, May 2000.

[Her00c] J. Herbst. A machine learning approach to workflow management. In *Machine Learning : ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain*, volume 1810, pages 183–194. Springer, May 2000.

[HK98] J. Herbst and D. Karagiannis. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. In *9th International Workshop on DEXA*, page 745. IEEE Computer Society, 1998.

[HK04] J. Herbst and D. Karagiannis. Workflow mining with inwolve. *Comput. Ind.*, 53(3) :245–264, 2004.

[Hol86] A. W. Holt. Coordination technology and petri nets. In *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory in Petri Nets-selected papers*, pages 278–296, London, UK, 1986. Springer-Verlag.

[JB96] S. Jablonski and C. Bussler. *Workflow Management : Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, 1996.

[Jen95] K. Jensen. *Coloured Petri nets : basic concepts, analysis methods and practical use, vol. 2*. Springer-Verlag, London, UK, 1995.

[JL96] S. Jarzabek and T.W. Ling. Model-based support for business re-engineering. *Information and Software Technology*, 38(5) :355–374, 1996.

[JVvdAR06] M. H. Jansen-Vullers, W. M. P. van der Aalst, and M. Rosemann. Mining configurable enterprise information systems. *Data Knowl. Eng.*, 56(3) :195–244, 2006.

[KEP00] G. Knolmayer, R. Endl, and M. Pfahrer. Modeling processes and workflows by business rules. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2000.

[KMO98a] B. Kiepuszewski, R. Muhlberger, and M. E. Orlowska. Flowback : providing backward recovery for workflow management systems. In *SIGMOD '98 : Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 555–557, New York, NY, USA, 1998. ACM Press.

[KMO98b] B. Kiepuszewski, Ralf Muhlberger, and Maria E. Orlowska. Flowback : providing backward recovery for workflow management systems. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 555–557. ACM Press, 1998.

[Kow92] R. Kowalski. Database updates in the event calculus. *J. Log. Program.*, 12(1-2) :121–146, 1992.

[KS86] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1) :67–95, 1986.

[KS95] N. Krishnakumar and A. P. Sheth. Managing Heterogeneous Multi-system Tasks to Support Enterprise-Wide Operations. *Distributed and Parallel Databases*, 3(2) :155–186, 1995.

[KS96a] F. N. Kesim and M. Sergot. A logic programming framework for modeling temporal objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(5) :724–741, 1996.

[KS96b] N. Kesim and M. Sergot. Implementing an object-oriented deductive database using temporal reasoning. *J. Database Manage.*, 7(4) :21–34, 1996.

[KT97] W. J. Kettinger and J. T. C. Teng. Business process change : a study of methodologies, techniques, and tools. *MIS Q.*, 21(1) :55–80, 1997.

[KT03] P. Kellert and F. Toumani. Les web services sémantiques. In *Web sémantique, Action spécifique 32 CNRS/STIC*, October 2003.

[LA94] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Syst. J.*, 33(2) :326–348, 1994.

[Law97] P. Lawrence. *Workflow handbook 1997*. John Wiley & Sons, Inc., 1997.

[LD99] F. Lindert and W. Deiters. Modelling inter-organizational processes with process model fragments. In Peter Dadam and Manfred Reichert, editors, *Enterprise-wide and Cross-enterprise Workflow Management*, volume 24 of *CEUR Workshop Proceedings*, pages 33–41. CEUR-WS.org, 1999.

[LR00] F. Leymann and D. Roller. *Production Workflow, Concepts and Techniques*. Prentice-Hall, Inc., 2000.

[MAGK95] C. Mohan, G. Alonso, R. Günthör, and M. Kamath. Exotica : A research perspective ob workflow management systems. *IEEE Data Eng. Bull.*, 18(1) :19–26, 1995.

[Mar02] D. C. Marinescu. *Internet-Based Workflow Management : Towards a Semantic Web*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[MC03] V. Miguel and F. Charoy. Bonita : Workflow cooperative system. http ://bonita.objectweb.org, 2003.

[Mic] Sun Microsystems. Java foudation classes (jfc/swing). http ://java.sun.com/products/jfc/.

[Mit95] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1995.

[MK94] R. Manganelli and M. Klein. Should you start from scratch ? *Management Review*, 83(2) :45–47, 1994.

[MKK+96] D. M. Min, J. R. Kim, W. C. Kim, D. Min, and S. Ku. Ibrs : intelligent bank reengineering system. *Decis. Support Syst.*, 18(1) :97–105, 1996.

[Mos81] E. B. Moss. Nested transactions : An approach to reliable distributed computing. Technical report, Cambridge, MA, USA, 1981.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1) :1–40, 1992.

[MRKS92] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A transaction model for multidatabase systems. In *ICDCS*, pages 56–63, 1992.

[MTV97] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3) :259–289, 1997.

[MWvdAvdB02] L. Maruster, A. J. M. M. Weijters, W. M. P. van der Aalst, and A. van den Bosch. Process mining : Discovering direct successors in process logs. In *5th International Conference on Discovery Science*, pages 364–373. Springer-Verlag, 2002.

[OS99] P. O'Neill and A. S. Sohal. Business process reengineering a review of recent literature. *Technovation*, 19(9) :571–581, 1999.

[OSS+97] A. Oberweis, R. Schätzle, W. Stucky, W. Weitz, and G. Zimmerman. Incomewf : A petri net based approach to workflow management. In *Wirtschatsinformatik97*, pages 82–101, Berlin, Germany, 1997. Springer-Verlag.

[PG04] S. S. Pinter and M. Golani. Discovering workflow models from activities' lifespans. *Comput. Ind.*, 53(3) :283–296, 2004.

[PG06] J. Vonk P. Grefen. A taxonomy of transactional workflow support. *International Journal of Cooperative Information Systems*, 15(1) :87–118, 2006.

[PR95] J. Peppard and P. Rowland. *The Essence of Business Process Re-engineering.* Prentice Hall, New York, 1995.

[PS94] D.P. Petrozzo and J.C. Stepper. *Successful Reengineering.* Van Nostrand Reinhold, New York, 1994.

[RGvdA+06] M. Rouached, W. Gaaloul, W. M. P. van der Aalst, S. Bhiri, and C. Godart. Web service mining and verification of properties : An approach based on event calculus. In *14th International Conference on Cooperative Information Systems (CoopIS'06)*, LNCS, Montpellier, France, November 1-3, 2006. Springer-Verlag.

[RS95a] A. Reuter and F. Schwenkreis. Contracts - a low-level mechanism for building general-purpose workflow management-systems. *IEEE Data Eng. Bull.*, 18(1) :4–10, 1995.

[RS95b] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. pages 592–620, 1995.

[RvdA05] A. Rozinat and W. M. P. van der Aalst. Conformance testing : Measuring the fit and appropriateness of event logs and process models. In *Business Process Management Workshops*, pages 163–176, 2005.

[SABS02] H. Schuldt, G. Alonso, C. Beeri, and H. J. Schek. Atomicity and isolation for transactional processes. *ACM Trans. Database Syst.*, 27(1) :63–116, 2002.

[Sch96] T. Schäl. *Workflow Management for Process Organisations*, volume 1096 of *LNCS*. Springer-Verlag, Berlin, 1996.

[Sch02] G. Schimm. Process Miner - A Tool for Mining Process Schemes from Event-Based Data. In *European Conference on Logics in AI*, pages 525–528. Springer-Verlag, 2002.

[Sch04] G. Schimm. Mining exact models of concurrent workflows. *Comput. Ind.*, 53(3) :265–281, 2004.

[SCSD02] M. Sayal, F. Casati, M.C. Shan, and U. Dayal. Business process cockpit. *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883, 2002.

[Sha90] M. Shanahan. Representing continuous change in the event calculus. In *ECAI*, pages 598–603, 1990.

[SKM+96] A. P. Sheth, K. Kochut, J. A. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shevchenko. Supporting state-wide immunisation tracking using multi-paradigm workflow technology. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 263–273. Morgan Kaufmann, 1996.

[SM01] A. Sharp and P. McDermott. *Workflow Modeling : Tools for Process Improvement and Application Development.* Artech House, Inc., Norwood, MA, USA, 2001.

[SR93] A. P. Sheth and M. Rusinkiewicz. On transactional workflows. *Data Engineering Bulletin*, 16(2) :37–40, 1993.

[Sun02] Sun. *Enterprise JavaBeansTM Specification, Version 2.1.* Sun Microsystems, August 2002.

[VBvdA01] H. M. W. (Eric) Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using woflan. *Comput. J.*, 44(4) :246–279, 2001.

[vdA98] W. M. P. van der Aalst. Three good reasons for using a petri-net-based workflow management system. *The Kluwer International Series in Engineering and Computer Science : Information and Process Integration in Enterprises : Rethinking Documents, Chapter 10*, 428 :161–182, 1998.

[vdA00a] W. M. P. van der Aalst. Workflow verification : Finding control-flow errors using petri-net-based techniques. In *Business Process Management*, pages 161–183, 2000.

[vdA00b] W.M.P. van der Aalst. Process design by discovery : Harvesting workflow knowledge from ad-hoc executions. Technical report, In M. Jarke, D.E. O'Leary, and R. Studer, editors, Knowledge Management : An Interdisciplinary Approach, Dagstuhl, July 2000. Dagstuhl Seminar Report, 281., 2000.

[vdA01] W. M. P. van der Aalst. Exterminating the dynamic change bug : A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3) :297–317, 2001.

[vdA05] W.M.P. van der Aalst. Process mining in cscw systems. In *Proceedings of the 9th IEEE International Conference on Computer Supported cooperative work in design (CSCWD 2005)*, pages 1–8, London, UK, 2005. Coventry University/IEEE Computer Society Press.

[vdAB01a] W. M. P. van der Aalst and T. Basten. Identifying commonalities and differences in object life cycles using behavioral inheritance. In *ICATPN '01 : Proceedings of the 22nd International Conference on Application and Theory of Petri Nets*, pages 32–52, London, UK, 2001. Springer-Verlag.

[vdAB01b] W. M. P. van der Aalst and P. J. S. Berens. Beyond workflow management : product-driven case handling. In *GROUP*, pages 42–51. ACM, 2001.

[vdABtHK00] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski. Advanced Workflow Patterns. In O. Etzion and Peter Scheuermann, editors, *5th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'00)*, number 1901 in LNCS, pages 18–29, Eilat, Israel, September 6-8, 2000. Springer-Verlag.

[vdAdCG+00] W. M. P. van der Aalst, P. J. N. de Crom, R. R. H. M. J. Goverde, K. M. van Hee, Wout J. H., H. A. Reijers, and R. A. van der Toorn. Exspect 6.4 : An executable specification tool for hierarchical colored petri nets. In Nielsen, M. and Simpson, D., editors, *Lecture Notes in Computer Science : 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark, June 2000*, volume 1825, pages 455–464. Springer-Verlag, 2000.

[vdAdM05] W. M. P. van der Aalst and A. Karla A. de Medeiros. Process mining and security : Detecting anomalous process executions and checking process conformance. *Electr. Notes Theor. Comput. Sci.*, 121 :3–21, 2005.

[vdARS05] W. M. P. van der Aalst, H. A. Reijers, and M. Song. Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6) :549–593, 2005.

[vdAS04] W. M. P. van der Aalst and M. Song. Mining social networks : Uncovering interaction patterns in business processes. In Jörg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 2004.

[vdAtHW03] W. M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business process management : A survey. In *Business Process Management*, pages 1–12, 2003.

[vdAvD02] W. M. P. van der Aalst and B. F. van Dongen. Discovering workflow performance models from timed logs. In *1st International Conference on Engineering and Deployment of Cooperative Information Systems*, pages 45–63. Springer-Verlag, 2002.

[vdAvDH+03] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining : a survey of issues and approaches. *Data Knowl. Eng.*, 47(2) :237–267, 2003.

[vdAvH02a] W. M. P. van der Aalst and K. M. van Hee. *Workflow Management : models, methods and tools.* Cooperative Information Systems. MIT Press, 2002.

[vdAvH02b] W.M.P. van der Aalst and K.M. van Hee. *Workflow management : models, methods and systems.* The MIT Press, 2002.

[vDdMV+05] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The prom framework : A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.

[vDvdA04] B. F. van Dongen and W. M. P. van der Aalst. Multi-phase process mining : Building instance graphs. In Paolo Atzeni, Wesley W. Chu, Hongjun Lu, Shuigeng Zhou, and Tok Wang Ling, editors, *ER*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 2004.

[VG03] J. Vonk and P. W. P. J. Grefen. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases*, 14(2) :137–172, 2003.

[vGW96] R. J. v. Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3) :555–600, 1996.

[vHSV89] K. M. van Hee, L. J. Somers, and M. Voorhoeve. Executable specifications for distributed information systems. In WG 8.1 Working Conference on Information System Concepts : An In-depth Analysis, Namur, Belgium, E. D. Falkenberg, and P. Lindgreen, editors, *Proceedings of the IFIP TC 8 , Elsevier Science Publishers, Amsterdam*, pages 139–156, 1989.

[VTKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1) :5–51, 2003.

[VvdA00] H. M. W. (Eric) Verbeek and W. M. P. van der Aalst. Woflan 2.0 : A petri-net-based workflow diagnosis tool. In *ICATPN*, pages 475–484, 2000.

[W3C00] W3C. *Simple Object Access Protocol (SOAP) V. 1.1.* W3C (World Wide Web Consortium), www.w3.org/TR/SOAP/, 2000.

[WFM95] WFMC. *Workflow Reference Model.* WFMC (Workflow Management Coalition), www.wfmc.org, January 1995.

[WR92] H. Watcher and A. Reuter. The contract model. pages 219–263, 1992.

[WS92] G. Weikum and H. J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In *Database Transaction Models for Advanced Applications*, pages 515–553. 1992.

[WS97a] D. Worah and A. P. Sheth. Transactions in transactional workflows. In *Advanced Transaction Models and Architectures*, pages 3–34. 1997.

[WS97b] D. Worah and A. P. Sheth. Transactions in transactional workflows. In *Advanced Transaction Models and Architectures*, pages 3–34. 1997.

[WvdA01] T. Weijters and W.M.P. van der Aalst. Process mining : Discovering workflow models from event-based data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 283–290, 2001.

[WvdA02] A. J. M. M. Weijters and W. M. P. van der Aalst. Workflow mining : Discovering workflow models from event-based data. In *ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 78–84, 2002.

[YW03] X. Yongyi and Z. Weishi. Component-based workflow architecture of a distributed software process management system. *qsic*, 00 :204, 2003.

[Zam94] H. Zampetakis. Survey re-engineering and rightsizing : Magic bullet often misses the target. *Financial Review*, page 38, 1994.

[zMM01] z. M. Muehlen. Process-driven management information systems - combining data warehouses and workflow technology. In Bezalel Gavish, editor, *Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR-4)*, pages 550–566, Dallas (TX), 2001. Southern Methodist University.

[ZNBB94] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In *ACM SIGMOD*, Minneapolis, USA, 1994.