Conception et construction de fédérations de progiciels

Hervé Verjus

THESE

présentée à

L'UNIVERSITE DE SAVOIE
ECOLE SUPERIEURE D'INGENIEURS D'ANNECY (ESIA)

par

Hervé VERJUS

pour obtenir le

DIPLÔME DE DOCTEUR DE L'UNIVERSITE DE SAVOIE

(Arrêté ministériel du 30 Mars 1992)

spécialité :

Informatique

Conception et construction de fédérations de progiciels

Soutenue publiquement le 24 septembre 2001 devant le jury composé de :

M. Jacky ESTUBLIER	Directeur de thèse	Directeur de Recherche au CNRS, Grenoble
M. Flavio OQUENDO	Directeur de thèse	Professeur à l'Université de Savoie, Annecy
M. Michel LEONARD	Rapporteur	Professeur à l'Université de Genève, Suisse
M. Michel RIVEILL	Rapporteur	Professeur à l'Université de Nice – Sophia-Antipolis, Nice
M. Yves CHIARAMELLA	Examinateur	Professeur à l'Université Joseph Fourier, Grenoble
M. Pierre-Yves CUNIN	Examinateur	Professeur à l'Université Joseph Fourier, Grenoble

Thèse préparée au sein du LLP/CESALP

Remerciements

Ce travail a été réalisé au sein du Laboratoire de Logiciels pour la productique (LLP/CESALP-ESIA) d'Annecy, en collaboration avec le laboratoire Logiciels, Systèmes, Réseaux (LSR-IMAG/UJF/CNRS) de Grenoble.

Je voudrais tout particulièrement remercier :

M. Michel Léonard et M. Michel Riveill pour avoir accepté d'être rapporteurs de cette thèse. Je leur suis reconnaissant pour leur lecture attentive ainsi que pour les critiques constructives qu'ils ont faites à ce travail.

M. Yves Chiaramella, pour m'avoir fait l'honneur de présider le jury de cette thèse.

M. Pierre-Yves Cunin, pour sa bienveillance tout au long de ces années et à bien des égards. Son action a été prépondérante tant au démarrage qu'à l'aboutissement de ce travail. Je le remercie pour avoir accepté d'être membre du jury de cette thèse, pour ses remarques et ses conseils ainsi que sa participation à la relecture du manuscrit.

M. Jacky Estublier, pour avoir dirigé ces travaux durant ces années de thèse et depuis mon DEA, malgré un certain éloignement géographique ; l'ensemble des conseils qu'il m'a prodigués et sa patience dans les « passages à vide », son expérience qui m'a été grandement profitable et les discussions que nous avons eues ont été d'une importance capitale.

M. Flavio Oquendo, pour m'avoir dirigé et pour m'avoir fourni un environnement propice à l'élaboration de cette thèse. Je le remercie pour ses nombreux conseils, sa disponibilité aux moments importants, son soutien dans de nombreux domaines et le témoignage sans cesse renouvelé de sa confiance.

J'adresse mes remerciements aux membres de l'équipe génie logiciel pour l'ambiance agréable, le soutien, l'aide et la bonne humeur. Un vif merci particulièrement à Ilham, à Sorana et à Thomas.

Je remercie M. Alain Haurat, directeur du LLP et tous les membres du laboratoire pour leur accueil et pour m'avoir fourni les moyens et l'ambiance nécessaires à l'aboutissement de ce travail.

Merci aux membres de l'équipe Adèle du laboratoire LSR pour leur soutien et leurs apports dans le cadre des travaux que nous avons menés.

Je salue les membres du projet européen PIE pour les critiques apportées à ce travail et les échanges fructueux.

Merci à Ilham et Valérie qui m'ont aidé dans la rédaction de cette thèse, avec leurs lectures et corrections minutieuses.

Merci à Valérie pour sa bonne humeur quotidienne (c'est tellement important !) et pour son soutien logistique.

A celles et ceux (dont l'anonymat sera préservé !), qui par leurs actions, leurs témoignages, leurs concessions (voire sacrifices), ont permis la réalisation de ce travail, qu'ils trouvent ici le témoignage de ma reconnaissance.

Abstract

Organizations aim at combining a lot of software applications together in wide and consistent systems. Such systems have to exhibit properties such as openness, interoperability, maintainability, autonomy, evolution, etc. Our approach, as in EAI (Enterprise Application Integration) domain, is aiming to interoperate (instead of integrate) autonomous, distributed, software applications and services that have to respond together at users needs.

We define a federation as a heterogeneous software tools set, most of these tools being commercial ones; such software tools are geographically distributed and autonomous.

Our approach deals with several facets that are mainly: conceptual facet, i.e. how one can build an information system composed of heterogeneous software tools; operational facet, i.e. how can one define the behaviour – control – of such system (the federation system); technological facet, i.e. how can one plug several software tools in a federation. We will show how the framework architecture we propose allows on the one hand to build software tools federations, and on the other hand to control their execution.

Keywords : Federations, Integration, Architecture, COTS, Software Tools, Interoperability

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

Chapitre I INTRODUCTION

I.1 Introduction à la problématique

L'informatique au sein des organisations durant ces dernières années a été marquée par l'apparition de technologies nouvelles alors que s'exprimaient des besoins de plus en plus importants des utilisateurs.

L'arrivée de grands progiciels de gestion intégrée, dotés de nombreuses fonctionnalités qui supportent les besoins des utilisateurs et la diversité des fonctions des entreprises, constituent le cœur des systèmes d'information. Parallèlement, les applications logicielles peuvent désormais communiquer, coopérer, grâce à l'émergence de nouvelles technologies permettant par la même occasion de franchir les portes et les frontières des organisations.

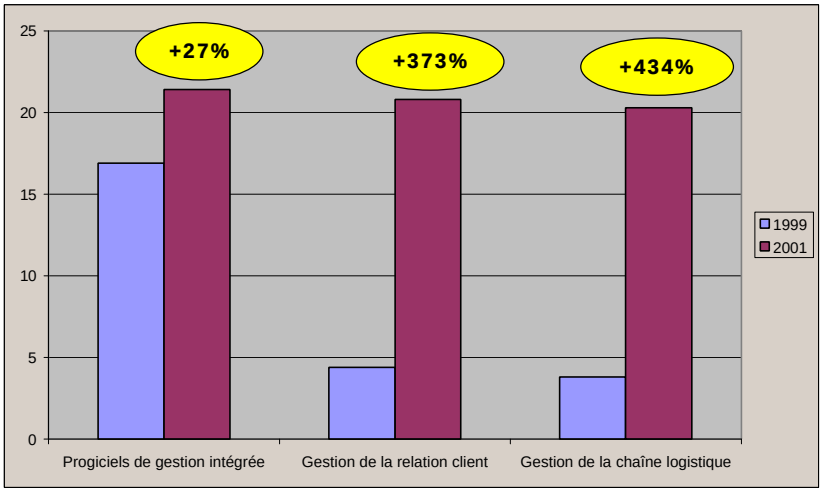

Figure I.1 Evolution du marché mondial des logiciels d'entreprises en milliards de dollars

(source: AMR Research, juin 2000)

Enfin, l'attrait pour Internet, passant du statut de simple média au rang de support "universel", et qui permet d'intégrer et d'offrir des services aux utilisateurs, d'avoir un rôle d'intégrateur d'applications...

La refonte des systèmes d'information des entreprises s'inscrit dans ce contexte, au milieu des paradoxes, des antagonismes, voire, des leurres technologiques.

Deux tendances se dessinent cependant et s'affrontent [ICSSEA 2001]. La première est de choisir un éditeur et de s'y tenir. Un éditeur important, dont le comportement parfois hégémonique et duquel dépendent les clients peut parfois conduire à l'impasse ou à des

situations de blocage. La seconde est de diversifier ses choix. Cette dernière s'accompagne généralement de contraintes techniques fortes.

Mis dans le contexte des applications d'entreprises (notamment les Progiciels de Gestion Intégré – PGI), un éditeur unique semble avoir beaucoup de mal à couvrir l'ensemble des fonctions métiers des entreprises et propose des applications souvent moins bonnes que les éditeurs, spécialistes métiers, et que les applications développées en interne. D'autre part, sachant que certains spécialistes de la relation client ou de la chaîne logistique ont des solutions adaptées aux besoins, les utilisateurs poussent les éditeurs de PGI à s'ouvrir.

C'est dans ce contexte, et dans le cadre des relations intra ou/et inter-entreprises que se situe notre problématique. Disposant d'outils divers, les groupements d'entités (entreprises réseaux, entreprises virtuelles, etc. [Favrel 1998, AMA 1991, Sandoval 1995]) tentent de mettre en place des systèmes d'information transversaux à la fois entre les services et entre les entreprises ; le but étant de pouvoir interconnecter un ensemble d'applications, un ensemble de services afin de répondre aux attentes des utilisateurs et aux nouveaux modes d'organisation [Reix 1986, Morton 1991, Reix 1995, Morton 1995] tout en gardant une certaine flexibilité. En effet, ces inévitables rapprochements, partages, communications, ne doivent pas se faire au détriment de l'autonomie de chaque unités, de chaque utilisateur, de chaque outil mis dans un certain contexte.

Forts de ces constatations, plusieurs groupes de travail se sont constitués ayant pour objectif de proposer des spécifications, des standards, des normes que ce soient sur les échanges informatisés, les technologies pour le Web et les bus logiciels. Pourtant, si certaines de ces propositions semblent s'imposer (XML par exemple), d'autres rentrent en concurrence et tendent à se complexifier (CORBA et Java/RMI, EJB, CCM, .NET, etc.).

De nombreux travaux se sont focalisés sur cette problématique : l'intégration d'outils. Cela part de la programmation classique où un programme est "découpé" en opérations, à la programmation orientée objet où on parle de classes, de paquetages à la prise en compte d'applications réparties. Différentes technologies ont été proposées pour mettre en synergie les applications et les utilisateurs [Fuggetta 1993, Totland et Conradi 1995] :

- les collecticiels,
- les outils de workflow,
- le génie logiciel,
- l'ingénierie des systèmes d'information,
- la modélisation d'entreprise,
- la conception des organisations,
- l'ingénierie concurrente,
- les Environnements de Génie Logiciel Centrés Processus.

Pourtant, peu de travaux ont mis l'accent sur l'assemblage d'applications hétérogènes et existantes, si ce n'est, ces dernières années, les approches dans le domaine de l'EAI (Enterprise Application Integration).

Malgré tout et comme nous le verrons dans les chapitres suivants, ces technologies n'ont pas permis de couvrir l'étendue des besoins, pas plus qu'elles ne permettent de s'adapter aux changements.

I.2 Pourquoi des fédérations ? Les hypothèses

L'idée de départ est de considérer que les applications logicielles de demain seront davantage des assemblages de composants logiciels déjà existants (prêts à l'emploi) aux seins des entreprises plutôt que de nouveaux produits issus de longs processus de développement. Nous partons du principe que la plupart des fonctionnalités requises existent déjà sous forme d'outils logiciels qu'il faut pouvoir intégrer.

I.2.1 L'autonomie

Nous nous intéressons aux systèmes qui sont construits à partir de composants existants. Ces composants sont majoritairement de deux types :

- les composants disponibles sur le marché (appelés "COTS"[1]),
- les composants disponibles et développés au sein des organisations (appelés composants, applications ou systèmes "patrimoines").

Dans le premier cas, les composants sont autonomes puisqu'ils peuvent s'exécuter en dehors du contexte du système qu'ils intègrent. Ils disposent, pour la plupart, de ressources locales, d'un modèle de fonctionnement interne (décrit par le code du composant).

Dans les deux cas, il est souvent intéressant d'adopter une approche de type "boîte noire" dans la mesure où :

- le code est indisponible pour les composants de type COTS ;
- le code est souvent difficile à maintenir, à faire évoluer pour les composants patrimoines.

Ainsi, nous partons du principe selon lequel nous ne pouvons pas, en règle générale, modifier le comportement intrinsèque du composant.

Ces composants sont donc :

> - autonomes,
> - hétérogènes,
> - distribués sur un réseau et potentiellement sur plusieurs sites géographiquement éloignés.

I.2.2 Une communauté... d'intérêts

L'objectif est de mettre ensemble des composants, des outils logiciels, pour permettre d'unifier les possibilités et les fonctionnalités des outils, et proposer un "tout" cohérent, aussi complet que possible au sein d'un système qui va permettre d'automatiser certaines tâches. La raison de la construction d'un tel système est de proposer aux utilisateurs une application (qui, par définition, propose un certain nombre de services), composée d'outils hétérogènes, dont le fonctionnement soit comparable (en termes de services rendus aux utilisateurs) à une application logicielle classique.

Sachant que les outils logiciels doivent ensemble satisfaire les objectifs de l'application, ils forment ce que nous appelons une organisation (une communauté d'intérêts). Cette organisation sera régie par un mode (ou politique, ou stratégie...) de fonctionnement qui

[1] Abréviation anglo-saxonne de "commercial off the shelf".

déterminera son comportement au cours du temps et donc, le comportement de chaque outil à l'intérieur de l'organisation.

Ainsi, les organisations de composants qui nous intéressent ont deux caractéristiques importantes qu'il faudra étudier :

1. les artefacts partagés (tout ou partie) de l'application ;
2. le fonctionnement commun du système.

I.2.3 Notion de fédération

Selon l'une des définitions proposée par "Le Petit Robert" [Robert 1989], une fédération est une *union d'entités, regroupées sous une autorité commune*. Sachant les hypothèses faites précédemment, le concept de fédération est parfaitement adapté aux organisations d'outils que nous voulons étudier. Ainsi, de telles organisations d'outils logiciels seront donc appelées dans la suite de ce manuscrit, des *fédérations d'outils* logiciels.

I.3 Cadre de la thèse

Les travaux de cette thèse ont eu lieu dans le cadre du projet ESPRIT IV LTR PIE (Process Instance Evolution).

Les partenaires de ce projet sont l'Université Joseph Fourier (Adèle/LSR-IMAG), l'Université de Savoie (LLP/CESALP), le Centre de Recherche Européenne de Xerox, Dassault Système (France), The Victoria University of Manchester, Teamware Group Ltd (Royaume Uni), Universitaet Dortmund (Allemagne) et Politecnico di Milano (Italie).

Le projet aborde le problème de l'évolution en-ligne des processus logiciels. L'évolution du processus est faite à travers un système de rétroaction, contenant des supports pour le monitoring, la prise de décision, le changement ainsi qu'un support pour la stratégie d'évolution [Cunin 2000, Alloui et al. 2000, Greenwood et al. 2000, Cîmpan 2000]. L'ensemble de ces outils, de ces systèmes forme ce que nous avons qualifié de fédérations d'outils dont le but (l'application) est l'évolution du processus sujet. Les travaux de cette thèse se placent dans la définition, la construction et la mise en oeuvre de la fédération d'outils [Estublier et Verjus 1999, Cugola et al. 2000b, Estublier et al. 2001a, Estublier et al. 2001b, Estublier et al. 2001c].

I.4 Organisation du document

Le document est organisé en trois parties. La première partie porte sur l'état de l'art selon deux axes : (1) le domaine des processus logiciels, leurs supports appelés environnements de génie logiciel centrés processus que nous étudierons particulièrement sous l'angle des fédérations, (2) les moyens et travaux existants par ailleurs qui, soit révèlent des similitudes avec nos travaux, soit qui permettent de résoudre certains aspects de notre problématique. En particulier, après avoir défini ce qu'est une "fédération d'outils", nous nous intéresserons aux approches par composants, aux mécanismes d'interopérabilité, aux approches EAI, etc.

La deuxième partie porte sur notre solution. L'approche que nous avons choisie pour construire des fédérations d'outils, les concepts que nous avons définis, ainsi que l'architecture sous-jacente font l'objet du quatrième chapitre.

La troisième partie contient la validation et l'implémentation de notre proposition. Elle contient le cinquième chapitre où seront présentés, d'une part un scénario montrant l'intérêt de l'approche et d'autre part l'implémentation du prototype permettant de construire et de mettre en oeuvre des fédérations d'outils. Une introduction au langage FTML (*Federation Tools Markup Language* – dérivé de XML) au travers d'un scénario appelé scénario de transfert de produit se trouve en fin de ce chapitre.

La conclusion propose une synthèse et un bilan du travail effectué durant cette thèse et un ensemble de perspectives liées à la continuation de ce travail, aux nouvelles applications ainsi qu'aux nouveaux thèmes de recherche.

Quatre annexes sont également fournis.

- L'Annexe A présente la grammaire du langage FTML rassemblée sous forme de documents de DTD (*Document Type Definition*).
- L'Annexe B contient, dans sa première partie, un certain nombre de classes Java appartenant au prototype. La seconde partie de cette annexe fournit un exemple d'un modèle de fédération d'outils basée sur un scénario repris dans le cadre du document de thèse et écrit en FTML. La troisième partie regroupe les traces générées par le prototype lors de nos expérimentations.
- L'annexe C contient une explication détaillée portant sur les mécanismes de contrôle des fédérations d'outils.
- L'Annexe D présente les publications réalisées dans le cadre de ce travail de thèse.

Chapitre II LES ENVIRONNEMENTS DE GENIE LOGICIEL CENTRES PROCESSUS ET LES PROCESSUS LOGICIELS

Les environnements de genie logiciel centres processus et les processus logiciels

Après une introduction sur les processus logiciels, nous proposons dans ce chapitre un état de l'art sur les environnements centrés processus logiciels selon deux points de vue : (1) le processus logiciel et (2) l'architecture du support. L'objectif est avant tout de voir leurs diversités, les problèmes auxquels ils sont confrontés, pour permettre de comprendre en quoi la direction prise au cours des dernières années n'a d'autres objectifs que de contourner les limitations des environnements de première génération. Cela nous servira à voir en quoi l'approche que nous proposons s'affranchit de certaines des limitations présentées à la fin de ce chapitre.

II.1 Les processus logiciels

II.1.1 Introduction

Bien que les fondements en matière d'automatisation remontent aux premiers langages de programmation, les technologies des processus logiciels n'ont vraiment connu un essor que ces trente dernières années avec l'arrivée d'outils et d'environnements de génie logiciel.

C'est à la fin des années soixante qu'un phénomène précipita les choses : la crise du logiciel. C'est alors que de vastes programmes furent entrepris dont les objectifs étaient d'améliorer la fabrication des logiciels que ce soit en termes de coûts, de délais, de qualité, de l'utilisation des technologies utilisées, de productivité c'est-à-dire, l'ensemble des paramètres entrant en jeu dans la chaîne de production des logiciels [Bourgeois 1985]. C'est ainsi qu'on vit apparaître le concept environnement de génie logiciel.

Peu à peu, la programmation prit de l'ampleur; à la fois par la taille des systèmes informatiques à développer mais aussi des moyens humains mis en jeu, qui étaient, au départ, limités ("programming in the small"), vers la réalisation de systèmes plus complexes ("programming in the large"), nécessitant davantage de moyens humains et technologiques ("programming in the many"). Ainsi, en réponse à la fois aux exigences de la complexification et de la quantité de logiciels à produire (besoin d'automatisation), mais également en réponse aux besoins de rationaliser la production de logiciels et d'accroître leur qualité, les premiers environnements de génie logiciel sont apparus [Humphrey 1991]. [Sommerville 1989] définit un environnement de génie logiciel comme étant un ensemble intégré d'outils et de mécanismes permettant de supporter toutes les phases de développement du logiciel (analyse, conception, écriture de code, test, etc.).

Des travaux se sont donc intéressés à définir, modéliser le logiciel et les étapes conduisant à sa fabrication, tant les facettes sont nombreuses (activités humaines, outils, automatisation, coopération, évolution, gestion des ressources et des produits, etc.). Parmi ces travaux sur les processus logiciels, on trouve les modèles de cycle de vie du logiciel, les environnements intégrés de génie logiciel et les environnements centrés processus logiciels.

II.1.2 Les processus logiciels

La manière dont le développement de logiciel est organisé, contrôlé, mesuré, supporté et amélioré (indépendamment du type de support technologique utilisé dans le développement) au sein d'une entreprise constitue le processus logiciel de cette dernière [Promoter 1999]. Même si elles exhibent des niveaux différents de sophistication dans la maîtrise de leur processus, toutes les organisations développant du logiciel suivent un processus, qu'il soit implicite ou explicite, reproductible, instrumenté, adaptable ou autre.

Un processus implique des personnes, des technologies, plus généralement un ensemble d'artefacts. Tout naturellement, un processus logiciel définit un ensemble d'artefacts propres au domaine du génie logiciel. On parlera d'activités, d'étapes, de tâches, d'acteurs, de participants, d'outils, etc. dont la sémantique est propre au domaine. Dans le domaine du logiciel, [Avrilionis 1996] a proposé un panel de définitions des termes couramment employés, tout comme [Conradi et al. 1992, Longchamp 1993] qui ont proposé des définitions génériques pour les concepts abordés dans le domaine des processus logiciels.

L'idée de processus (ou procédé) est liée au fait que l'humain résout les problèmes auxquels il est confronté en créant des descriptions de processus appelés modèles de processus. Il décline, pour chaque problème particulier un processus particulier à partir d'une description qui se veut générique : il s'agit d'une instance. Puis, chaque instance peut être exécutée [Osterweil 1987].

Il est bien entendu que, pour la plupart des organisations attachées à leurs méthodes de travail et leurs visions de leurs processus, les processus mis en œuvre doivent être rationalisés le plus possible (réduction des coûts, possible rétroaction, etc.). Tout l'intérêt est donc de définir explicitement et aussi finement que possible les artefacts qui le composent afin de pouvoir reproduire, avec un maximum de généricité, le processus ainsi défini.

II.1.3 Les modèles de cycle de vie

Ce sont les premiers travaux visant à caractériser le processus logiciel, c'est-à-dire à décrire l'enchaînement des étapes de la conception à la maintenance d'un produit logiciel : son cycle de vie. Les modèles de cycle de vie les plus connus sont ceux dits "en cascade" [Royce 1970] et "en spirale" [Boehm 1981, Boehm 1986]. Ces modèles ont permis de mieux comprendre le processus logiciel par la description des activités abstraites et de leurs ordonnancement. Cependant ces modèles ne tiennent pas compte de l'activation, de l'échec ou au contraire du succès des activités et, en général, de la synchronisation des activités entre elles, pas plus qu'ils ne s'intéressent aux artefacts manipulés par ces activités (les ressources, l'organisation, etc.) [Madhavji 1991, Curtis et al. 1992, Huff 1996].

Toute autre information nécessaire à la compréhension du processus qui ne peut être décrite par ces modèles fait généralement l'objet d'une description informelle à l'intérieur d'un document accompagnant le modèle.

II.1.4 La maîtrise des processus, la qualité du logiciel

Les processus logiciels, bien que différents suivant l'organisation qui va les définir et les mettre en oeuvre, n'en possèdent pas moins un caractère générique. La qualité du logiciel obtenue n'est pas seulement liée au produit, mais également à l'organisation et au processus de production qui est employé [Derniame et al. 1994]. Des méthodes ont été proposées favorisant la compréhension générale du processus et permettant aux organisations qui mettent en oeuvre des processus logiciels de mieux les maîtriser, de les évaluer afin de pouvoir les améliorer (ce qui améliore le produit logiciel résultant).

Ainsi, plusieurs approches permettant de mesurer la maturité du processus ont été proposées : le *Modèle d'Evaluation des Capacités du Logiciel* (en anglais *Capability Maturity Model* ou *CMM*) développé par *Software Engineering Institute* au Carnagie-Mellon University [Paulk et al. 1991] et ses contreparties européennes telles que *Bootstrap* [Kuvaja 1994] et *SPICE* [Rout 1995] supportés par *European Software Institute*, à Bilbao. Ces efforts reflètent une évolution du concept de qualité du logiciel vers des environnements centré-processus. Derrière cette évolution se trouve la conviction que l'amélioration de la qualité ainsi que la réduction des coûts sont mieux servies en certifiant des processus et en faisant en sorte que ces processus soient suivis.

L'ensemble de ces méthodes n'ont d'autre but que d'évaluer les processus mis en oeuvre au sein des organisations et de proposer des actions pour l'amélioration des processus.

II.1.5 Les environnements intégrés de génie logiciel

Se sont succédés différents travaux, Make [Feldman 1979], Smaltalk [Goldberg 1984], Interlis [Teitleman et Masinter 1984], etc. Cependant et malgré le succès de ce genre d'environnements, la dimension processus n'est pas réellement abordée dans le sens où ni la définition, ni l'enchaînement des activités ne sont proposées (la façon dont le produit logiciel est fabriqué n'est pas explicité).

C'est dans la perspective d'assister les développeurs dans leurs tâches qu'une nouvelle génération d'outils est apparue : les environnements intégrés de génie logiciel [Longchamp et al. 1992]. Il s'agissait en fait de proposer aux développeurs, sous une forme homogène et intégrée, un ensemble de fonctionnalités nouvelles reposant sur l'intégration d'outils souvent existants et connus des développeurs (compilateurs, éditeurs de liens, gestionnaires de fichiers, etc.). L'aspect intégré permettait d'utiliser cette collection d'outils dans la perspective de produire des logiciels avec davantage de facilités.

Pour remédier aux limitations des environnements intégrés, et dans le but de prendre en compte le processus logiciel, la deuxième génération d'environnements sera présentée dans la section consacrée aux *Environnements de Génie Logiciel Centrés Processus (EGLCP)* [2].

[2] On utilisera les expressions "environnement centré processus" [Snowdon et Warboys 1994] ou encore "environnement de génie logiciel centré processus" ou encore "environnement de support au processus" sans distinction sémantique dans ce document dans la mesure où notre démarche ne porte pas sur le processus directement mais sur son support.

II.2 Les environnements de génie logiciel centrés processus

II.2.1 Qu'est-ce qu'un EGLCP ?

Les approches et travaux présentés dans la section précédente révèlent une limitation majeure : soit ils proposent une méthodologie rigide (modèles de cycle de vie), soit ils intègrent des outils, des technologies logicielles sans méthodologie précise.

L'un des objectifs majeurs des *Environnements de Génie Logiciel Centrés Processus* est justement de fournir aux utilisateurs à la fois certaines des technologies de production du logiciel tout en intégrant une partie de la gestion de cette dernière dans un environnement relativement intuitif.

Au cours de ces dix dernières années, de nombreux travaux se sont focalisés sur les environnements centrés processus logiciels : Marvel [Ben-Shaul et Kaiser 1993, Kaiser et al. 1988], Oz [Ben-Shaul et Kaiser 1995, Ben-Shaul et Kaiser 1998], Arcadia [Taylor et al. 1988], Peace [Arbaoui 1993, Arbaoui et al. 1992, Arbaoui et Oquendo 1993a, Arbaoui et Oquendo 1993b, Arbaoui et Oquendo 1994], Alf [Canals et al. 1994], Scale [Oquendo 1995], Peace+ [Alloui et al. 1994, Alloui et Oquendo, 1996a, Alloui 1996], OIKOS [Ambriola et al. 1990, Ambriola et Montangero 1992, Montangero et Ambriola 1994], EPOS [Conradi et al. 1994], SPADE [Bandinelli et al. 1992, Bandinelli et al. 1993a, Bandinelli et al. 1993b, Bandinelli et al. 1994a], JIL/Little JIL [Sutton et al. 1997, Sutton et Osterweil 1997, Lerner et al. 1998], Tempo [Belkhatir et Melo 1992, Belkhatir et al. 1993, Belkhatir et Melo 1993, Melo 1993], APEL [Dami et al. 1995, Estublier et al. 1998a], etc. La plupart des EGLCP développés offre un langage de modélisation pour la définition du processus (que l'on appelle *Langage de Modélisation de Processus*[3]) et un environnement d'exécution pour interpréter les modèles définis à partir de ce langage.

Le modèle du processus

Un modèle de processus est une abstraction du processus logiciel décrit de manière formelle ou semi-formelle.

Un modèle de processus comprend [Bandinelli et al. 1994b] :

- la description des ressources (outils, acteurs, etc.) que requiert le processus ;
- les activités et les tâches pour lesquelles le processus est définit et structuré ;
- l'enchaînement (ou ordonnancement) de ces activités ou tâches ;
- les informations nécessaires à la définition du processus.

Le modèle de processus fixe les fondements nécessaires à la coopération et à la communication entre les différentes entités participant au processus. Plusieurs sous-modèles ont été identifiés [Promoter 1999], qui ne font pas tous l'objet de consensus :

- un modèle d'activité exprimant les activités simples et complexes (agrégation d'activités simples) décrit à l'aide d'un formalisme particulier ;
- un modèle de produit exprimant les données qui sont manipulées par les activités ;

[3] Le terme anglophone fréquemment rencontré dans la littérature est PML (*Process Modelling Language*).

- un modèle d'outils pour décrire les outils utilisés dans le cadre du processus et leurs architectures. Cela peut être réalisé également par le modèle d'activités, en considérant l'outil comme l'enveloppe d'une activité ;
- un modèle organisationnel qui décrit la structure et contrôle les activités ainsi que leurs ressources ;
- un modèle utilisateur exprimant la manière dont les différents acteurs tirent parti de l'assistance fournie au travers du support technologique du processus, leurs responsabilités dans le contexte du processus, etc.

Cette description diffère légèrement de celle proposée dans [Promoter 1999]. Nous attribuons cette différence au fait que ces modèles concernent différentes vues du processus sous-jacent. Les modèles peuvent se chevaucher dans les artefacts qu'ils utilisent/exploitent ou, au contraire, se compléter. La question cruciale qui demeure est de gérer la cohérence entre l'ensemble des vues d'un même processus.

II.2.2 Les différents formalismes

Plusieurs approches ont été à l'origine de formalismes différents. Entre autres nous notons les langages de programmation classiques, les formalismes de modélisation comme les réseaux de Pétri, etc. Parmi les approches et les langages de modélisation de processus qui en découlent on peut noter [Amiour 1999] :

- l'approche procédurale illustrée dans les systèmes ARCADIA [Osterweil 1987, Sutton et al. 1995b], ProcessWise [Bruynooghe et al. 1994], etc.
- l'approche déclarative illustré par les systèmes Marvel [Kaiser et al. 1988, Barghouti et Krishnamurthy 1993a], Merlin [Junkerman et al. 1994], etc.
- l'approche orintée but adoptée par l'environnement Peace [Arbaoui et Oquendo 1993a, Arbaoui et Oquendo 1993b, Arbaoui et Oquendo 1994] et Peace+ [Alloui et al. 1994, Alloui et Oquendo 1996a, Alloui et Oquendo 1996b], SCALE [Oquendo 1995].
- l'approche basée sur les réseaux de Pétri illustré par les systèmes Process Weaver [Fernstrom 1993, Fernstrom et Brnoisz 1993], SPADE [Bandinelli et al. 1994a] et Leu [Dieters et Gruhn 1998].
- l'approche fonctionnelle illustré par les systèmes HFSP [Suzuki et Katayama 1991] et à PDL [Inoue et al. 1989], etc.
- l'approche mutli-paradigme prise en compte dans les systèmes comme Alf [Canals et al. 1994], APEL [Dami et al. 1995, Estublier et Dami 1995, Amiour 1997, Amiour et Estublier 1997, Estublier et al. 1998a], etc.

II.2.3 Formalisation du processus

On distingue trois phases principales dans le cycle de vie d'un processus logiciel [Ambriola et al. 1997] :

- la phase de spécification des besoins qui consiste à identifier et/ou documenter les besoins du processus. On pourra, par exemple, définir la performance attendue, les objectifs, etc. ;
- la phase de conception, de modélisation qui doit définir le processus pour répondre aux besoins exprimés lors de la phase précédente ;

- la phase d'implantation du processus tel qu'il a été définit.

Afin de répondre aux objectifs du processus et de chacune de ces trois phases et ainsi, de couvrir le cycle de vie, des langages ont été proposés selon trois catégories que l'on retrouve dans [Ambriola et al.. 1997] :

- un langage de spécification répondant aux besoins de la phase de spécification, dont les concepts doivent avoir pour but de décrire l'organisation, les objectifs généraux, etc. ;
- un langage de modélisation pour modéliser le processus : c'est ce langage qui devra couvrir, pour l'essentiel, les concepts de base du processus logiciel ;
- un langage d'implantation dont l'objectif sera de décrire la façon dont doit être exécuté le processus.

Il est acquis que les formalismes de haut niveau sont à privilégier afin de fournir des moyens adaptés (simples) pour la description du processus [Sutton et al. 1995a, Jaccheri et al. 1998]. En effet, le but du domaine n'est pas d'alourdir la compréhension du processus ainsi décrit mais bien de la simplifier et de permettre aux différents acteurs (manager, développeurs, etc.) d'avoir une vision uniforme du processus.

Idéalement, un formalisme de modélisation doit satisfaire les besoins suivants [Oquendo 1995] :

- il doit être exécutable,
- il doit permettre de décrire et de supporter l'ensemble du cycle de vie du processus ainsi que tous ses niveaux d'abstraction,
- il doit prendre en compte la description dynamique de l'ordonnancement des activités du processus,
- il doit permettre de supporter et de modéliser l'évolution des processus et de leurs modèles,
- il doit permettre de modéliser et de gérer l'incertitude et l'incomplétude de certaines informations,
- il doit permettre d'exprimer et de supporter la communication, la coordination, la négociation et la coopération entre les divers intervenants dans le processus.

Sachant que le but est donc d'assister le concepteur et le développeur dans la compréhension et dans la modélisation du processus, des langages graphiques, semi-formels et de haut niveau ont été développés. C'est ainsi que les langages de modélisation tels que OOA/OOD [Coad et Yourdon 1991a, Coad et Yourdon 1991b], OMT [Rumbaugh et al. 1991], UML [Muller 1996], etc. ont largement inspiré certains formalismes de description de processus (APEL [Estublier et al. 1998a], Process Weaver [Fernström 1993, Fernström et Brnoisz 1993], etc.). Ces langages viennent en complément de langages formels, indispensables pour décrire de façon rigoureuse les modèles de processus.

Les différents domaines du processus logiciel

Les processus ont ainsi pu, grâce aux EGLCP, bénéficier de supports pour ce qui concerne leur modélisation, et leur exécution. Même si ces environnements proposent des approches différentes, l'objectif demeure identique à savoir supporter des processus logiciels. Aussi, ont-ils des caractéristiques identiques. C'est dans [Dowson et Fernström 1994, Promoter 1999] que l'on trouve une distinction entre les concepts génériques inhérents à l'ensemble des

EGLCP et l'identification des besoins liés aux mécanismes d'exécution. C'est ainsi que le processus logiciel est structuré en trois domaines distincts (voir Figure II.1) : le domaine de définition de modèles de processus[4], le domaine d'exécution de modèles de processus[5] et le domaine de la mise en œuvre du processus[6]. Cette distinction permet de mieux comprendre le processus logiciel en général ainsi que les besoins liés aux EGLCP, car comme il a été montré, la concordance entre les trois domaines n'est pas toujours au rendez-vous [Cîmpan 2000].

> Le **domaine de la définition** contient des descriptions de processus ou fragments de processus exprimés dans une certaine notation définissant ce qui pourrait ou bien devrait être mis en œuvre. Une telle description du processus est appelée *modèle du processus*.

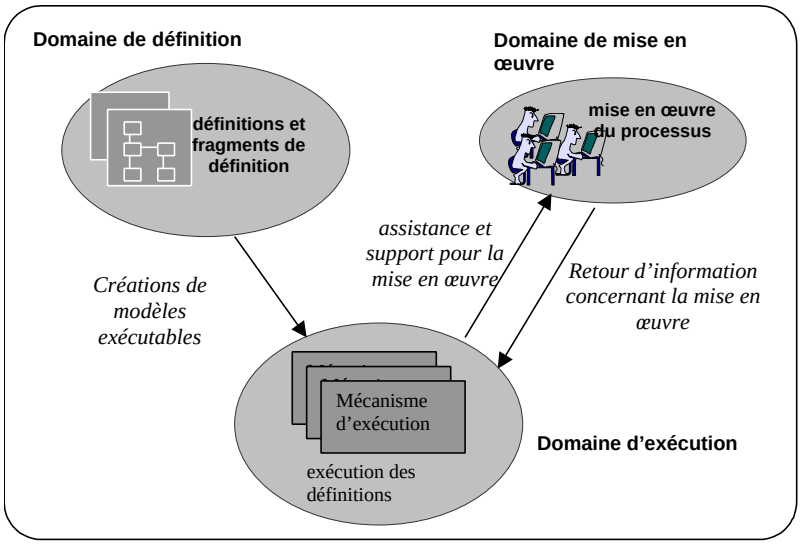

Figure II.1 Les trois domaines du processus (d'après Dowson et Fernström)

Dans le cas des EGLCP cette notation est "exécutable", c'est-à-dire elle peut être interprétée par un mécanisme d'exécution.

> Le **domaine de la mise en œuvre** englobe les activités effectives ainsi que les actions prises en charge par tous les participants, concernant aussi bien les activités de développement que celles de gestion du processus.

Une partie de ces activités seront conduites utilisant des outils logiciels, d'autres seront complètement en dehors du support, comme par exemple des interactions entre les participants au processus.

> Le **domaine d'exécution** est concerné par ce qui se passe dans un EGLCP pour supporter le processus mis en œuvre, admettant que ce dernier est gouverné par la définition qui a été donnée ; il interprète le modèle du processus à partir d'une représentation exécutable et englobe les mécanismes et les ressources nécessaires de l'environnement pour supporter les domaines de définition et de mise en œuvre.
>
> Le *mécanisme d'exécution* utilise les définitions afin de déterminer ses interactions avec les participants impliqués dans la mise en œuvre ainsi qu'avec d'autres composants de l'environnement, le but final étant d'assurer l'aide, le guidage ou bien le contrôle de la

[4] La désignation anglophone utilisée est "Process Definition Domain"
[5] La désignation anglophone utilisée est "Process Enactment Domain"
[6] La désignation anglophone utilisée est "Process Performance Domain"

> mise en œuvre en vue d'assurer sa cohérence avec la définition du processus qui a été donnée.

En d'autres termes, tandis que le processus est mis en œuvre dans le monde réel (domaine de la mise en œuvre), il est exécuté par la machine dans le monde abstrait du modèle de processus (domaine de l'exécution).

II.3 Vers des EGLCP fédérés

II.3.1 La réutilisation de composants dans les EGLCP

La section précédente a introduit les principaux concepts du domaine des processus logiciels et les principales caractéristiques des EGLCP. Certains travaux du domaine se sont intéressés à des aspects particuliers comme la distribution de l'environnement, la coopération, la coordination des composants, l'intégration de composants selon une approche non-intrusive, etc. D'autres ont défini des EGLCP comme des fédérations de composants qui doivent inter-opérer.

II.3.1.1 Leu

LEU [Graw et Gruhn 1995] est un environnement basé sur les réseaux Funsoft. Le processus peut être exécuté par plusieurs moteurs de processus. Dans ce cas, chacun des moteurs va exécuter des fragments de modèles (ou bien sous-modèles). En ce sens, le processus est donc lui-même distribué sur plusieurs moteurs de processus formant ainsi un environnement global.

LEU est un environnement homogène dans le sens où l'ensemble des moteurs partagent les concepts et le formalisme décrivant les processus. Il est possible d'ajouter des moteurs de processus à l'environnement qui vont pouvoir exécuter des modèles (ou fragments de modèles).

II.3.1.2 Peace, Peace+

Peace [Arbaoui et al. 1992, Arbaoui 1993, Arbaoui et Oquendo 1993a, Arbaoui et Oquendo 1993b, Arbaoui et Oquendo 1994] est un EGLCP qui adopte une approche orientée but fondée sur une logique non-monotone. L'approche orientée but se concentre sur la description (basée sur la logique des croyances de [Moore 1988]) des buts à atteindre dans un processus plutôt que sur la description des activités.

Peace décompose le processus en fragments de processus représentant une étape du processus ainsi qu'un but à atteindre. Peace connut deux extensions, chacune d'entre elles reposant sur une approche multi-agents : (1) Peace+ [Alloui et al. 1994, Alloui 1996, Alloui et Oquendo 1996a, Alloui et Oquendo 1996b] dont l'objectif est d'améliorer les aspects liés aux travaux dans les environnements coopératifs ; (2) Peace* [Latrous et Oquendo 1995, Latrous et Quendo 1996, Latrous 1997] dont l'objectif est d'améliorer la gestion de l'évolution des processus.

Peace est un environnement homogène ; le processus global est distribué au niveau des différents agents de l'environnement. Il permet également à d'autres PSS[7] d'interopérer avec lui en utilisant le format d'échange WPDL [WfMC 1996b].

[7] On trouve dans la littérature le terme anglophone "Process Sensitive-Software" pour "systèmes sensibles aux processus".

II.3.1.3 Provence

Provence [Barghouti et Krishnamurthy 1993a, Barghouti et Krishnamurthy 1993b] est un EGLCP reposant sur une approche client/serveur selon laquelle les clients peuvent s'abonner aux évènements qui sont produits durant l'exécution du processus. Le but de cet environnement est de reposer sur des outils existants. Il peut être vu, dans une certaine mesure, comme une généralisation de Marvel en terme d'interopérabilité et d'intégration d'outils. Cette dernière est assurée par un composant de l'architecture: le gestionnaire de mise en œuvre et son rôle consiste à faire inter-opérer les outils logiciels participants à l'environnement.

Provence a été implémenté avec les composants suivants :

- Marvel [Ben-Shaul et Kaiser 1993] qui gère la cohérence, la disponibilité des objets manipulés au sein de l'environnement, gère la coordination et la coopération entre les différents acteurs (personnes au sein du même processus logiciel) et permet la définition du modèle de processus en termes de règles ;
- Yeast assure la liaison avec les clients et déclenche les actions correspondantes aux requêtes des utilisateurs ;
- 3D File System fournit un mécanisme permettant aux utilisateurs de créer des vues dynamiques du logiciel et d'effectuer des changements relatifs à cette vue sans affecter la base du logiciel ;
- Doty, éditeur graphique, qui permet de créer et de manipuler les graphes de façon interactive en utilisant le multi-fenêtrage. L'interprétation des graphes se fait par Doty et le langage qui lui est associé.

Provence pourrait être vu comme une fédération d'outils dans le sens où il repose sur une volonté de proposer un environnement de "composants inter-opérables". En ce sens, Provence offre une architecture relativement indépendante des outils qui le composent, contrairement à la plupart des environnements existants où les outils sont fortement imbriqués, ne pouvant être remplacés. Pourtant si les outils sont bien identifiés et ne sont pas fortement intégrés dans l'environnement, le phénomène d'ajout ou de retrait d'outils implique une forte modification de l'interface entre les composants et doit tenir compte des spécificités liées aux composants d'origine. En effet, certaines hypothèses ont été faites selon les possibilités de ces outils. Par exemple, Marvel doit connaître les chemins des différents fichiers composant le projet et doit savoir quels outils utilisés sur quels objets (cette information étant "codée en dur").

D'autre part, l'architecture proposée s'appuie sur les possibilités intrinsèque de Marvel, Yeast, Doty, 3D File System. Le remplacement de ces outils par d'autres devrait se faire qu'à la condition que les nouveaux outils disposent des mêmes fonctionnalités et répondent aux mêmes caractéristiques d'implémentation que ceux utilisés. Ces contraintes n'étant pas décrites dans l'environnement (aucune description des contraintes et des caractéristiques des composants), le respect de ces dernières reste au bon vouloir du concepteur de l'environnement. On peut également noter que l'environnement, tel qu'il a été conçu, ne peut pas fonctionner si l'un ou l'autre des outils est absent.

II.3.1.4 Endeavour

Le système Endeavours [Bolcer et Taylor 1996] est un système distribué dont le but est de supporter les processus logiciels et les processus de workflow. L'originalité de Endeavours repose sur une approche homogène pour la modélisation des processus et l'approche hétérogène pour leurs exécutions.

Cela signifie qu'un modèle de processus est définit selon plusieurs modèles basés sur des concepts identiques et décrits à l'aide d'un même formalisme mais qu'ensuite, chacun de ces modèles peut être interprété sur des plate-formes différentes: en effet, il existe des "handlers", écrits en langage Java, qui sont l'implémentation des modèles.

Les "handlers" selon Endeavours constituent les composants de l'environnement à l'exécution. En ce sens, il ressemblent aux fragments de processus décrits dans l'environnement Peace+.

II.3.2 Vers des fédérations de composants inter-opérables pour les EGLCP

Les environnements précédents reposent tous sur des architectures à base de composants qui doivent inter-opérer en vue de réaliser les objectifs décrits par les modèles de processus sous-jacents. Certaines de ces approches ont davantage porté sur la distribution du processus au niveau de plusieurs moteurs d'exécution (Alf, Scale, Leu, Peace+), d'autres ont mis l'accent sur les problèmes d'hétérogénéité (Endeavour), d'autres encore considèrent un environnement à partir de composants existants et hétérogènes (Provence).

En revanche, aucun de ces systèmes n'aborde réellement la définition des fédérations en considérant l'existence de "règles communes de fonctionnement" [Verjus et Oquendo 1998, Estublier et Verjus 1999] et ne fournissent des moyens pour la modélisation des fédérations.

Les systèmes qui vont suivre mentionnent explicitement la définition de "fédération".

II.3.2.1 Oz

Le projet Oz a étudié l'interopérabilité entre environnement de génie logiciel centrés processus; en particulier [Ben-Shaul et Kaiser 1995] porte sur l'interopérabilité d'EGLCP et les différentes possibilités notamment en ce qui concerne l'hétérogénéité de modèles de processus. Les mêmes auteurs ont poursuivi leurs travaux sur des propositions d'architectures [Ben-Shaul et Kaiser 1998] et, en particulier, ont proposé une architecture pour des fédérations d'EGLCP selon une perspective homogène :

- les composants de la fédérations sont tous des environnements Marvel pouvant être potentiellement géographiquement distribués ;
- le comportement de la fédération est régit selon la métaphore de *l'Alliance Internationale* selon laquelle les composants peuvent constituer des "sommets" à partir desquels il en découle des "traités" qui vont régir le fonctionnement de l'environnement Oz. Chaque environnement garde son autonomie dans la mise en œuvre de ses processus. Cependant, il leur est possible d'interagir pour la réalisation d'une activité commune (le traité). Les sommets n'expriment donc que les stratégies de mise en œuvre des traités; dans un sommet, un environnement coordonne l'activité commune: (1) il reçoit les données nécessaires à l'activité et provenant des autres environnements participant au sommet, (2) il met en œuvre l'activité et (3) il transmet les résultats aux autres environnements. Ce comportement est obtenu par l'implémentation des procédures et protocoles du sommet comme mécanismes de base de chacun des environnements de la fédération.

L'approche consiste donc à distribuer l'exécution des modèles de processus sur des sites différents et en fonction de la disponibilité des ressources nécessaires à leurs exécutions.

L'architecture proposée est relativement flexible dans le sens où elle permet l'ajout et le retrait de composants Marvel de façon dynamique.

L'une des principales limitations est le caractère homogène de l'approche (modèles de composants et formalisme utilisé) et le mode de fonctionnement (la métaphore) qui ne peut être changé.

II.3.2.2 Modéliser des fédérations d'EGLCP

L'approche prise dans [Tiako 1999] porte sur des fédérations de fragments de processus dérivés de LCPS [Derniame et al. 1994]. Des modèles de contrats entre environnements peuvent alors être constitués permettant la délégation de modèles, etc. Il s'agit donc d'une approche permettant de modéliser les fédérations sous l'angle de l'organisation des environnements la composant et supportant la mobilité de ces derniers. Chaque contrat définit également les compétences respectives de chaque EGLCP impliqué dans ce dernier.

En fait, l'approche prise considère les compétences d'un EGLCP sous la forme des services qu'il peut offrir et les possibles négociations permettant à un EGLCP de déléguer l'exécution d'un modèle ou fragment (appelé aussi composant de processus) de modèle de processus à un autre EGLCP.

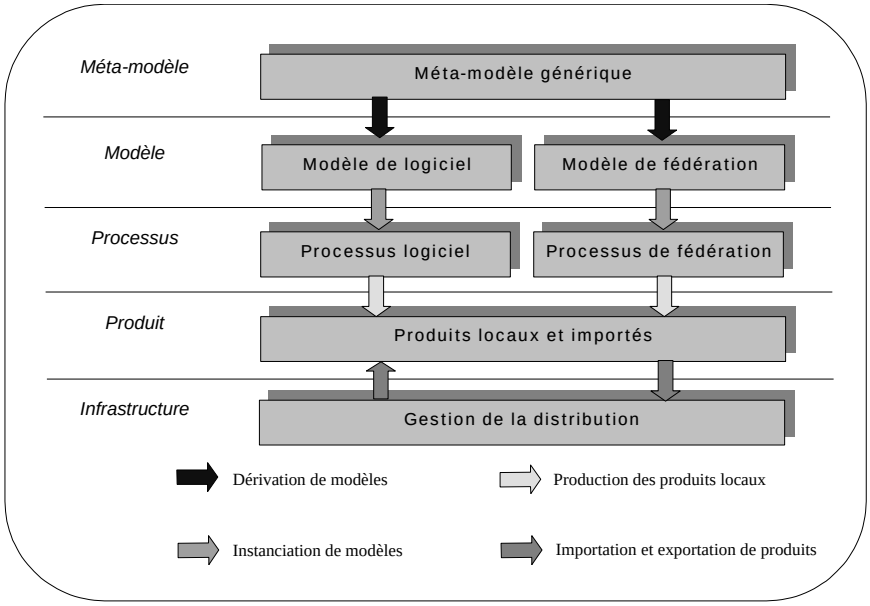

Figure II.2 Architecture pour des fédérations EGLCP

[Tiako 1998] propose alors une architecture à cinq niveaux comprenant (voir figure II.2) :

1. un méta-modèle qui permet à la fois de décrire des modèles de processus mais également des modèles de fédération (d'EGLCP) ;
2. un niveau modèle regroupant un ensemble de modèles de processus logiciels et de processus de fédération, basés sur les concepts du méta-modèle ;
3. des processus logiciels d'une part et des processus de fédération d'autre part sont obtenus comme instances des modèles (respectivement de processus et de fédération) ;
4. un couche produit qui contient l'ensemble des produits qui sont produits et importés par en EGLCP. Ces produits peuvent être échangés entre EGLCP grâce au niveau infrastructure ;
5. une couche intermédiaire (infrastructure) entre les constituants permet alors les échanges et la communication.

L'originalité de cette proposition tient au couplage des processus logiciels et des processus de fédérations rendu possible par le partage de nombreux concepts (comportement, similarités

des concepts de rôle et de compétences, des concepts d'agents et de participants,…) [Tiako 1998, Tiako 1999]. D'où l'idée de recourir à un méta-modèle générique permettant de définir les processus de fédérations au même niveau d'abstraction que les processus logiciels.

L'approche constitue une alternative à Oz par les protocoles de négociations, de délégations et la mobilité des composants de processus. En revanche, il faut définir les types de base (en concepts dérivés de LCPS) qui seront utilisés pour modéliser les processus partagés par les différents EGLCP.

II.3.2.3 APELv4 et PIE

L'approche prise par [Amiour 1999, Estublier et al. 1999] consiste à proposer un "environnement coopératif et ouvert dans lequel le support se fait à travers un ensemble de composants. Chaque composant prend un charge la gestion d'un aspect particulier du procédé". Un tel environnement est qualifié de *fédération de composants interopérables*. L'environnement APEL v4 [Amiour et al. 1997] est la continuité des travaux effectués sur l'environnement APEL [Estublier et al. 1997a, Estublier et al. 1997b, Estublier et al. 1998a].

La justification de l'adjectif coopératif se trouve dans les mécanismes d'interaction entre les composants qui collaborent à la réalisation du processus. Quant à l'adjectif ouvert il est justifié si on considère qu'il est toujours possible d'ajouter un composant de l'environnement, sous réserve du respect de certaines contraintes (voir plus loin dans cette section).

Les composants qui sont pris en compte ici respectent un certain schéma : ils sont de véritables logiciels sensibles aux processus (LSP) offrant :

- un méta-modèle qui contient les aspects pertinents liés à l'aspect du processus auquel le composant est dédié ;
- des formalismes permettant de décrire l'aspect en question ;
- une plate-forme d'exécution permettant d'exécuter les modèles description des descriptions. Cette plate-forme comprend, entre autre, un moteur d'exécution, une base de données pour le stockage des informations, etc.

Cette description des composants qui sont pris en compte est restreinte à une catégorie de composants (LSP) et ne peut être généralisée. Par contre, tout logiciel sensible aux processus, qu'il soit un composant issu du marché ou un système patrimoine peut faire partie des fédérations décrites selon cette approche, sous réserve d'adhérer au méta-modèle d'APEL.

L'architecture conceptuelle de la fédération comprend cinq parties (voir figure II.3) [Amiour 1999] :

1. un modèle commun ;
2. un méta-modèle commun et un langage commun ;
3. une fondation ;
4. un ensemble de composants ;
5. une infrastructure de communication.

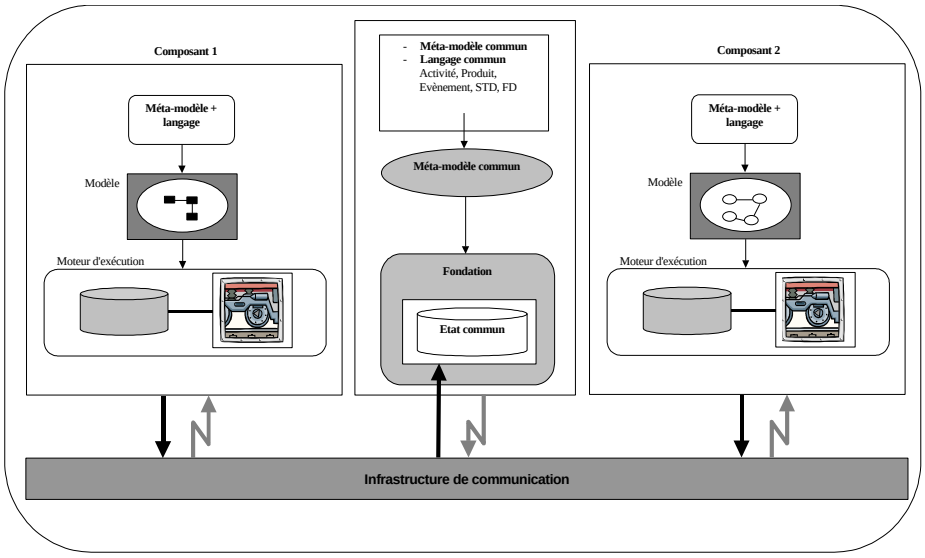

Figure II.3 Architecture conceptuelle de la fédération [Amiour 1999]

Le modèle commun permet de décrire les parties communes entre les composants et, entre autre :

- les activités et les produits qui font partie du processus à gérer (aspects statiques et dynamiques) ;
- les flux de contrôle (ordonnancement des activités) ;
- les flux de données (échanges des produits entre les activités).

Le modèle commun constitue donc, dans une certaine mesure, un accord et un objectif commun de la fédération. Une fois décrit, la fédération doit donner les moyens de garantir son exécution de manière cohérente; l'entité en charge de la cohérence de l'exécution est appelée la fondation. Une instance du modèle commun en exécution est appelée l'état commun constituée d'objets et de relations représentant les activités, les produits, etc. Cet état est sous le contrôle de la fondation.

Le modèle commun doit être décrit de façon a pouvoir être interprété (compris) par l'ensemble des composants (hétérogènes). Ces derniers utilisent les éléments du modèle pour définir leurs propres modèles (locaux). Dans cette perspective, le méta-modèle commun et le langage commun ont pour objectif de permettre la description du modèle commun :

- le méta-modèle commun fournit une ontologie[8] commune permettant aux divers composants de se comprendre tout en préservant leur autonomie. Il définit une sémantique pour l'ensemble des concepts ;
- le langage commun est un formalisme compréhensible par les composants et permet de matérialiser le méta-modèle commun et de décrire le modèle commun de la fédération.

[8] Une ontologie est définie comme une spécification explicite d'une conceptualisation [Gruber 1993] ou comme un "accord sur une conceptualisation partagée par une communauté" [Dieng et al. 2000]. Une ontologie fournit un cadre unificateur pour réduire et éliminer les ambiguités et les confusions conceptuelles et terminologiques et assurer une compréhension partagée par la communauté visée [Uschold et Gruninger 1996]. Pour représenter un domaine donné, il est nécessaire de se restreindre à un certain nombre de concepts significatifs suffisants pour interpréter ce domaine [Valente et Breuker 1996]. Une ontologie se fonde sur certains engagements ontologiques permettant de choisir un ensemble donné de concepts plutôt qu'un autre. Une ontologie comporte des définitions fournissant le vocabulaire conceptuel permettant de communiquer au sujet d'un domaine ; cela permet de définir (a) les concepts utilisables pour décrire la connaissance, (b) les relations entre de tels concepts et (c) leurs contraintes d'utilisation [Dieng et al. 2000].

Le rôle principal de la fondation est de coordonner les différents composants. Elle peut être considérée comme un serveur d'état enrichie du modèle commun. Elle :

- garantit l'exécution cohérente d'une instance du modèle commun par l'interprétation du modèle. Elle comprend donc un moteur d'exécution capable d'interpréter le modèle ce qui constitue une différence par rapport à un serveur d'état "simple" ;
- elle reçoit et traite les requêtes manipulant l'état commun qui émanent des composants. A cet égard, elle gère les problèmes de persistance, reprise après panne, etc. ;
- elle notifie les composants des changements intervenants dans l'état commun au travers d'un mécanisme de souscription / publication.

Les composants doivent quant à eux respecter certaines spécifications (méta-modèle, modèle et exécution). Un composant n'est pas obligé d'accepter l'ensemble des concepts du méta-modèle mais uniquement ceux qu'il est amené à utiliser. Il est doté d'une interface lui permettant de communiquer avec l'infrastructure de communication.

L'infrastructure de communication est en charge :

- de l'acheminement des requêtes (des composants vers la fondation) ;
- de l'acheminement des notifications (de la fondation aux composants).

PIE

Le projet européen PIE (*Process Instance Evolution*) a pour objectif d'étudier l'évolution des instances de processus, les problèmes de déviations et de concordance [Cîmpan et Oquendo 1998, Cîmpan 2000, Cîmpan et Verjus 1998a, Cîmpan et Oquendo 2000a, Cîmpan et Oquendo 2000b], de réconciliation [Cîmpan et Verjus 1998b], d'implémentation des changements [Alloui 1998] selon une ou des stratégies d'évolution [Roberston 1998] et de fournir un environnement, support aux processus et aux processus d'évolution.

Le serveur de processus de l'environnement APEL a servi de composant pivot dans l'environnement de PIE [Amiour et Estublier 1998b] dans le sens où (voir figure II.4) :

- il est la fondation de l'environnement PIE, l'univers commun [Estublier et al. 1998b, Estublier et Verjus 1999], l'état global de la fédération ;
- il permet aux autres composants (composant de monitoring, composant de décision, composant d'implémentation des changements, composant gérant la stratégie d'évolution, etc.) de participer (d'inter-opérer) à l'environnement grâce à une couche de communication.

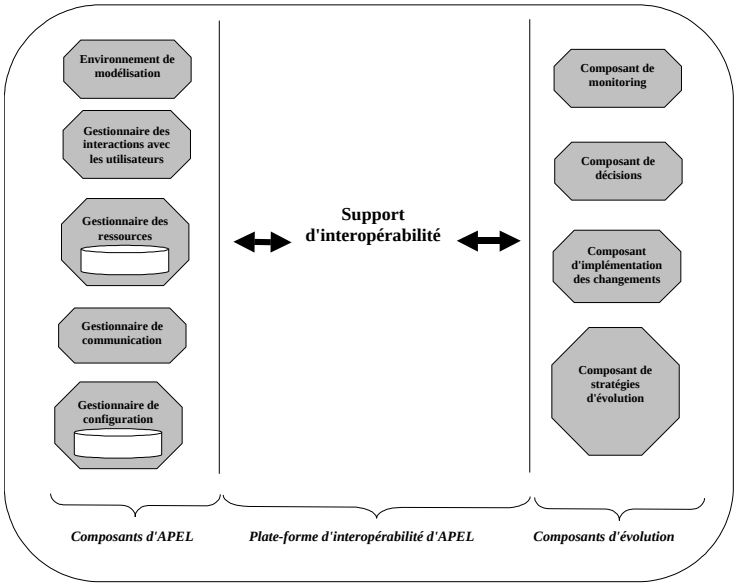

Figure II.4 Architecture de l'environnement PIE (d'après [Amiour et Estublier 1998b])

Les composants étaient "libres" d'observer le processus en s'abonnant aux sujets qui les concernaient et en recevant les notifications associées (notifications envoyées par la fondation). D'autre part, tout composant pouvait opérer certains changements sur l'état courant des instances du processus, en invoquant les services de l'interface (API) de la fondation.

II.4 Bilan et conclusion

Les environnements dans le domaine des processus logiciels ont eu pour principal objectif de répondre aux besoins immédiats dans le domaine du développement de logiciel, en proposant une vision "unifiée" des outils utilisés.

Cependant, rares sont les environnements ouverts et flexibles permettant à des outils de type COTS de participer à un environnement tout en restant disponibles par ailleurs (préservation de l'autonomie).

II.4.1 Les apports du domaine des EGLCP

On peut noter quelques apports du domaine des processus logiciels :

- le guidage du développement de logiciels grâce aux processus logiciels : les technologies des processus logiciels permettent de guider le comportement d'un environnement et des acteurs de ce dernier en vue de réaliser une application logicielle parfois complexe ;
- la formalisation des processus : les approches et les technologies développées permettent de structurer des processus de développement par l'identification de concepts, la définition de modèles basés sur ces derniers et la définition d'instances ;
- la mise en synergie de fonctionnalités hétérogènes pour satisfaire les besoins des développeurs : les derniers travaux portant sur les EGLCP ont mis en évidence une demande grandissante en fonctionnalités diverses, de mettre à disposition des développeurs une "boîte d'outils adaptable" dont il est possible de définir un certain mode de fonctionnement et, entre autre, permettre l'automatisation de certaines tâches.

II.4.2 Les limitations

Les principales limitations des EGLCP sont les suivantes :

- complexité, redondance et insuffisance des supports : les EGLCP sont des environnements de plus en plus complexes car ils intègrent de nombreuses fonctionnalités. Cependant, certaines sont manquantes alors que d'autres sont redondantes avec des fonctionnalités proposées par des composants existants sur le marché ;
- rigidité, monolithisme et faible évolution des supports : les EGLCP sont des environnements qui sont générés à partir d'une description. L'évolution de tels supports impose l'évolution de la description. Toute modification de la description induit une re-génération de l'environnement qui ne peut se faire en cours d'exécution ;
- faible hétérogénéité des constituants : les EGLCP "modulaires" intègrent rarement des composants hétérogènes ;
- faible autonomie des constituants : l'intégration de composants est forte et ne permet pas à ces derniers de préserver leur autonomie et de pouvoir être utilisés par ailleurs ;
- peu (pas) d'EGLCP qui intègrent des composants quelconques issus du marché (COTS) : les outils intégrés par les EGLCP sont bien souvent de type "load and go".

II.4.3 Conclusion

Comme nous l'avons vu dans la première partie de ce chapitre, le domaine des processus logiciels est vaste et les processus logiciels sont intrinsèquement complexes. De nombreuses recherches poursuivent leurs efforts pour caractériser, pour mieux cerner les processus : des approches ont été proposées pour les modéliser et des environnements ont été développés pour les supporter.

Dans le domaine du développement des logiciels, les approches et les technologies évoluant rapidement, de nouveaux outils apparaissent fréquemment, intégrant eux mêmes de nouvelles fonctionnalités. Les utilisateurs (acteurs des processus logiciels) sont de plus exigeants, les besoins continuent de croître et font de plus en plus appel à des outils logiciels spécifiques qu'il faut pouvoir faire inter-opérer tout en préservant la flexibilité et l'évolution des environnements. Plusieurs travaux ont porté sur l'intégration, la coopération, l'inter-opérabilité de composants logiciels au sein des EGLCP [Estublier et al. 1992, Belkhatir et al. 1994, Alloui et Oquendo 1996a, Di Nitto et Fuggetta 1995, Ben-Shaul et Kaiser 1998, Estublier et Barghouti 1998, Lerner et al. 1998] en vue de fournir un environnement complet [Alloui et al. 1999] Nous constatons cependant qu'aucune proposition ne satisfait simultanément aux critères [Verjus 1998a, Verjus et Oquendo 1998] :

- d'hétérogénéité,
- de distribution,
- de flexibilité/ouverture,
- d'autonomie,
- de contrôle.

De plus, la plupart des fonctionnalités requises au niveau d'un EGLCP peuvent être fournies par des outils logiciels issus du marché. Cette tendance vient s'opposer aux environnements générés et homogènes existants.

Ainsi, les environnements doivent :

- pouvoir intégrer des outils hétérogènes et distribués de façon non-intrusive ;
- garantir l'autonomie des outils qu'ils intègrent ;
- garantir le fonctionnement global et la cohérence de l'environnement composé de différents outils, comme s'il s'agissait d'une application développée spécifiquement pour des besoins propres.

Il s'agira donc de proposer des environnements flexibles, adaptables, ouverts, permettant de faire du "sur mesure".

En outre, les supports existants reposent essentiellement sur une approche qui considère l'outil logiciel comme simple ressource ou alors comme acteur d'une activité : il n'est pas identifié comme un concept majeur et sa dimension est réduite alors que paradoxalement, les besoins sont nombreux et la participation des outils logiciels ne cesse de croître (objectifs d'automatisation et de productivité). Les problèmes bien connus qui se posent comme le recouvrement de concepts entre outils, la cohérence des modèles opérationnels des différents outils, la gestion de la synchronisation, etc. sont totalement occultés.

Nous orientons notre approche vers une vision d'assemblage d'outils, de constituants, que nous appelons une *fédération d'outils* dont l'objectif est de construire des environnements à partir de progiciels existants qui doivent coopérer.

Notre approche est de considérer cette fédération d'outils logiciels selon trois axes :

1. les constituants: les composants ou outils ;
2. la (politique de) coordination et de contrôle de ces outils définissant le fonctionnement du support logiciel et garantissant sa cohérence, tout en préservant l'autonomie de chaque outil (approche non intrusive) ;
3. l'évolution (l'adaptation) du support aux besoins.

Chapitre III FEDERATION D'OUTILS : CADRE, TERMINOLOGIE ET ETAT DE L'ART

Fédération d'outils : cadre, terminologie et etat de l'art

L'objectif de ce chapitre est l'étude des fédérations d'outils, de composants et surtout, permettre de recenser certains travaux pertinents permettant de construire les fédérations d'outils. Nous adresserons donc des travaux connexes à notre problématique et autres travaux dont les objectifs présentent quelques similarités avec notre démarche.

III.1 Besoins pour les fédérations

III.1.1 Introduction

Le chapitre précédent a mis certains besoins en évidence, notamment par le fait que les utilisateurs ont des demandes de plus en plus difficiles à satisfaire d'une part, mais que d'autre part, certaines de ces demandes sont potentiellement traitées par des composants existants du marché (cas des composants COTS). Ainsi, il serait intéressant de disposer d'un environnement permettant l'intégration d'outils logiciels de type COTS ou des composants dits patrimoines en vu de réaliser une application logicielle. L'idée directrice étant de pouvoir construire un environnement modulable selon l'approche suivante : en partant de l'expression des besoins, nous identifions les composants qui permettent, ensemble, de satisfaire ces besoins et nous assemblons ces composants en vue de construire l'environnement spécifique.

III.1.2 Construire des fédérations

Comme nous le verrons dans ce chapitre quelques communautés de recherche étudient certains aspects connexes à la problématique que nous voulons traiter. Pour construire des fédérations, les points suivants sont à prendre en compte :

1. la description des composants de la fédération ;
2. l'assemblage de ces composants (techniques d'assemblage et paradigmes de fonctionnement) ;
3. les architectures offrant un support aux fédérations (architectures permettant de mettre en oeuvre des fédérations d'outils).

Ces points vont être au cœur des travaux présentés dans le cadre de ce chapitre.

III.1.3 Pourquoi une nouvelle architecture ?

Comme nous l'avons vu dans le domaine des processus logiciels, les EGLCP qui nous ont été proposés sont limités par certains aspects. Entre autre :

- ils ne sont pas exhaustifs dans l'éventail des fonctionnalités qu'ils proposent et qui sont requises par le processus logiciel,
- ils ne sont pas "bâtis" sur des composants existants,
- ils n'offrent aucune facilité à l'intégration de composants de type COTS,
- ils sont rigides dans la stratégie de fonctionnement (quand une stratégie est définie explicitement).

Ainsi, les architectures des différents environnements sont des architectures figées, difficilement adaptables.

Parallèlement et dans d'autres domaines, les environnements à forte composante logicielle qui sont proposés aux utilisateurs révèlent des faiblesses sensiblement identiques alors qu'au contraire, comme nous l'avons montré, la tendance est à l'intégration d'outils, d'applications logicielles autonomes en vue de mettre à la disposition des utilisateurs un environnement complet :

- disposant de l'ensemble des fonctionnalités requises,
- adaptable/évolutif,
- qui permet l'intégration de composants existants selon l'approche "boîte noire",
- dont le fonctionnement peut être changé selon les besoins.

Ce chapitre est donc consacré à l'étude des différentes approches portant sur des systèmes "bâtis" et reposant sur l'assemblage de composants et les méthodes ou moyens permettant cet assemblage. Aussi, nous aborderons successivement le domaine vaste de l'ingénierie des systèmes à base de composants, l'intégration des applications d'entreprise, les travaux consacrés à l'interopérabilité et à la coordination de composants et enfin, les systèmes fédérés incluant les bases de données fédérées et les systèmes fédérés de workflow.

III.2 L'ingénierie des systèmes à base de composants

Nous allons présenter dans cette section, les approches marquantes dans les approches à "composants". En particulier, nous étudierons les systèmes à base de composants "classiques", les architectures logicielles et les systèmes à base de composants issus du marché (appelés COTS).

III.2.1 Le phénomène "composant" et les systèmes à base de composants

Au cours de ces dernières années, plusieurs travaux ont porté sur les approches orientées composants. Il y a eu des propositions émanant d'entreprises (SUN, Microsoft, etc.), de groupes de recherches (architectures logicielles, serveurs d'application, intégration d'applications d'entreprises, etc.), d'organismes de normalisation (OMG). Nous allons, dans la suite de cette section, regarder brièvement les différentes propositions qui composent l'éventail des "visions" des composants ; nous retiendrons que l'idée transversale aux diverses propositions est de construire des applications (voire méta-applications), à partir de composants ou applications existantes et de faciliter les tâches des développeurs dans ce cadre (minimiser l'écriture de code).

III.2.1.1 Des "briques" de base...aux composants

Initialement, les langages de programmation se sont intéressés à la décomposition d'un programme en sous-programmes ou sous-parties ; nous avons vu plusieurs tendances opérer des découpages dont le découpage fonctionnel est sans doute le plus connu. Ainsi, petit à petit nous avons vu apparaître les termes de procédure (Pascal), de fonction (C, C++), de "regroupements" de fonctions en bibliothèques (C, C++), en paquetages (Ada, Java), etc. Ce phénomène s'est amplifié avec l'arrivée de la programmation orientée-objet et les concepts de modularité et d'encapsulation.

Ainsi, les programmes (ou applications logicielles) font souvent appel à ce qu'il est commun d'appeler des librairies, des bibliothèques. Dès lors, des fonctions, méthodes, procédures étaient regroupées au sein d'entités certes autonomes (DLL, etc.) et pouvant être réutilisées mais non exécutables seules...Cette approche ayant connu un succès considérable, elle s'est petit à petit généralisée passant du terme "bibliothèque" au terme "composant", vu comme une entité logicielle possédant certaines propriétés, suivant un cycle de vie. Parmi ces composants, nous trouvons les composants ActiveX, les contrôleurs OCX/OCL, etc [Chappell 1996b]. Ces composants sont tous des petits composants n'ayant rien à voir avec les composants que nous étudions, de part la taille et les fonctionnalités qu'ils sont censés fournir. Pour la plupart, il s'agit de composants graphiques qu'il est intéressant d'intégrer dans une application logicielle lors de la phase de développement de cette dernière.

Dans la suite, le concept de composant s'affranchit de certaines contraintes, reposant pour la plupart sur des spécifications émanant de grands constructeurs, éditeurs de logiciels.

III.2.1.2 La vision des composants

Dans [Orfali et al. 1996] il est écrit qu'un composant est "un morceau de logiciel assez petit pour qu'on puisse le créer et le maintenir, et assez grand pour qu'on puisse l'installer et en assurer le support. De plus, il est doté d'interfaces standard pour pouvoir interopérer."

Cependant, la définition de composants logiciels que nous trouvons le plus souvent prend sa source dans l'approche orientée objet qu'elle étend à la notion de composant par [Orfali et al. 1996] :

- c'est une ***entité commercialisable*** : c'est un logiciel autonome prêt à l'emploi, qu'il est possible d'acheter dans le commerce ;
- ce ***n'est pas une application complète*** : il doit être combiné à d'autres composants pour former une application complète (il dispose d'un nombre limité de fonctionnalités dans un domaine d'application particulier) ;
- il peut être utilisé en ***agencements non prévus*** à l'avance : comme les objets, il peut être utilisé dans des combinaisons que le développeur n'avait pas prévues à l'avance ;
- il possède une ***interface définie*** au travers de laquelle il peut être manipulé (analogie avec l'objet) ;
- c'est un ***objet qui sait interopérer*** : "un composant peut être appelé comme objet au travers d'espaces d'adresse, de réseaux, de langages, de systèmes d'exploitation, et d'outils variés. C'est une entité logicielle indépendante des systèmes[9]" ;
- c'est un ***objet*** au sens où il supporte l'encapsulation, l'héritage et le polymorphisme.

[9] L'expression "objet interopérable" a été utilisée par Ray Valdes dans un numéro spécial du Dr Dobb's Journal sur les objets interopérables (janvier 1995).

Cette définition est assez représentative des approches du domaine de l'ingénierie des systèmes à base de composants (CBSE)[10] où le composant n'existe que dans le but d'être une partie d'un composé ; il n'a donc pas d'intérêt à exister seul (existence intrinsèque).

Nous allons regarder certaines approches dans le domaine du CBSE qui déclinent cette vision du composant.

III.2.1.3 Les "Java Beans"

Les objectifs de SUN concernant les "*Java Beans*" [Javasoft 1997] sont de permettre la programmation par composition dynamique. Pour cela, les *beans*, sont l'appellation de composants réutilisables (visuels ou non) pour les plates-formes Java.

L'approche *Java Beans* se limite à la construction d'applications clientes (c'est pour cela qu'elle est bien adaptée à la construction d'interfaces graphiques) mais ne traite pas de la répartition des composants. La communication entre les *beans* se fait par évènements (selon le paradigme des abonnements/publications). La spécification des *beans* décrit quelques conventions (en particulier d'ordre syntaxique) :

int setX(int i) et int getX()

qui identifient un attribut X de type entier avec la convention de nommage "set" pour changer la valeur de l'attribut et "get" pour lire cette valeur.

La persistance n'est pas traitée explicitement dans la spécification *Java Beans*. Ce point est laissé à l'initiative du développeur qui devra recourir au mécanisme de sérialisation du langage Java si ce point doit être traité. Le développeur peut choisir quelles seront les propriétés qui devront être sauvegardées.

En fait, cette spécification délimite un cadre de développement de composants "clients", à la fois dans "l'esprit" Java et pour les plate-formes Java.

III.2.1.4 Les "Enterprise Java Beans"

Tout d'abord, précisons une chose importante, à savoir que les "*Enterprise Java Beans*" (EJB) ne doivent pas être confondus avec les "*Java Beans*" présentés précédemment.

L'objectif de la spécification *Enterprise Java Beans* (car il s'agit d'une spécification) est de définir une architecture permettant de construire une application en Java dont la partie serveur est construite à partir de composants appelés "*Enterprise Beans (EB)*" [Sun 1999]. En ce sens, cette vision orientée "serveur" est différente de l'approche cliente (*Java Beans*). Les composants EB sont écrits indépendamment de la plate-forme sur laquelle ils vont être exécutés ("écrire une fois, s'exécuter partout"). Selon l'approche EJB, l'architecture est considérée comme étant un ensemble d'interfaces.

L'architecture EJB identifie les éléments suivants [Patzer 2000] :

- composants logiciels ou *beans* (EB),
- conteneurs,
- serveurs,
- clients.

[10] L'abréviation anglophone couramment utilisée est CBSE (Component-Based System Engineering) que nous garderons comme telle dans le reste du document.

Les conteneurs isolent les *beans* de leurs clients et d'une implémentation spécifique d'un serveur. Les conteneurs et les serveurs implémentent les mécanismes de bas niveau utilisés dans les applications (transactions, persistance, gestion de la mémoire, etc.).

La spécification EJB définit :

- des rôles associés aux différentes personnes intervenant dans la production d'une application (assembleur d'application, fournisseur, utilisateur, administrateur) ;
- des contrats associés à un *bean* (ces contrats sont passés entre le conteneur et les clients qui utilisent le *bean*) ; ce sont des règles (obligations) qui doivent être respectées par le fournisseur de l'EB et le conteneur.

Les applications complexes en Java deviennent, avec cette approche, plus "simples" à écrire puisque plusieurs aspects (transactions, persistance, sécurité, répartition) sont déjà gérées. D'autre part, la plate-forme et le bus logiciel sont indépendants des applications.

III.2.1.5 Le modèle de composant COM/DCOM

L'approche Microsoft est finalement assez semblable aux approches que nous trouvons par ailleurs. Il s'agit de fournir un modèle de composants (en l'occurrence COM) puis un modèle de composants distribués (DCOM) utilisés par les développeurs pour développer des composants qui vont pouvoir interopérer sur les plates-formes MS-Windows. Le modèle de composant impose des contraintes technologiques assez lourdes (environnement homogène, langages supports, et structure binaire surtout) pour le développeur, mais permet cependant de faire interopérer des composants et produits logiciels fournis par des éditeurs différents. COM [COM 1995] définit une interface programmable (API) permettant aux divers composants, créés à partir de cette interface, d'être intégrés dans des applications existantes ou d'interagir.

COM est une spécification et une implémentation associée à cette dernière permettant à des clients d'invoquer des services fournis par de composants respectant la spécification (appelés objets COM). Les objets COM sont exposés au travers d'un ensemble d'interfaces constituant l'unique point de contact entre les client et l'objet. Les objets COM et leurs interfaces sont spécifiés en utilisant le langage de description d'interface de Microsoft (MIDL pour Microsoft Interface Description Language), langage dérivé du langage de définition d'interface de DCE[11]. Ainsi, les objets COM et les clients peuvent être codés dans n'importe quel langage supportant la structure binaire de COM/Microsoft.

La principale différence entre COM et DCOM est que le premier permet de faire interopérer des composants à l'intérieur d'une même machine (mais dans des espaces d'adressage différents) ; les extensions fournies par DCOM [DCOM 1997] permettent l'interopérabilité de composants respectant les spécifications au travers d'un réseau, pour autant que les services DCOM soient disponibles sur l'ensemble des machines sur lesquelles s'exécutent les composants.

[11] DCE (Distributed Computing Environment est un ensemble intégré de services au sein d'un environnement distribué dont le but est de permettre l'interopérabilité d'applications logicielles hétérogènes et distribuées sur le réseau. Pour plus de détail sur DCE : [OSF 1996, Schill 1993, Chappell 1996a].

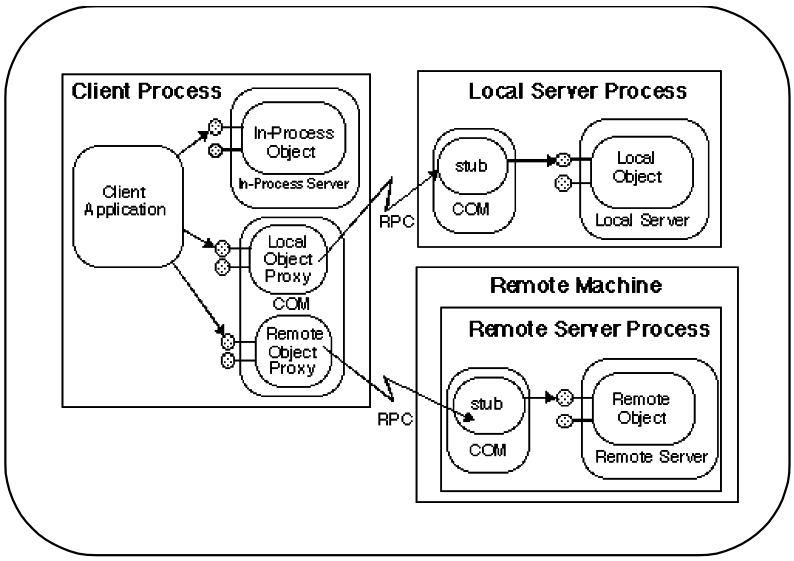

Figure III.1 Les objets COM sont accessibles selon trois méthodes [COM 1995]

Chaque objet COM s'exécute à l'intérieur d'un serveur, lui-même capable de contenir plusieurs objets. Il y a trois manières distinctes (voir Figure III.1), pour un client, d'accéder aux objets COM fournis par un serveur :

1. un client peut établir un lien direct avec une bibliothèque (ou librairie) contenant le serveur. Le client et le serveur s'exécutent à l'intérieur d'un même processus (in-process). La communication entre le client et le serveur se fait au travers d'appels de fonctions ;
2. le client accède au serveur s'exécutant dans un processus différent mais sur la même machine au travers d'un mécanisme de communication inter-processus. Ce mécanisme est basé sur le protocole RPC ("Remote Procedure Call" pour appel de procédure à distance) ;
3. le client accède au serveur distant s'exécutant sur une autre machine en utilisant le protocole RPC/DCE. Le mécanisme supportant l'accès aux serveurs distants est appelé DCOM.

Dans les faits, COM et DCOM sont combinés à l'intérieur d'une "couche d'exécution"[12] qui fournit les services d'accès locaux et distants. COM et DCOM fournissent des services de bas niveau ; OLE [Brockschmidt 1995] et ActiveX [Chappell 1996b, Active 1997] représentent les services de plus haut niveau construit sur les couches COM et DCOM. OLE [Chappell 1996b] est utilisé pour la construction d'objets liés et "embarqués" à l'intérieur de documents composites et ActiveX étend OLE pour permettre aux composants d'être inclus dans des sites Web. Par contre, l'approche DCOM permet de gérer les transactions avec les mécanismes proposés dans MTS (Microsoft Transaction System).

III.2.1.6 Objets métiers CORBA et modèle de composant CORBA

La spécification CORBA 2 était destinée à l'interopérabilité de composants en milieux hétérogènes [OMG 1991]. Plusieurs produits commerciaux implémentant cette spécification ont vu le jour, avec un certain succès d'ailleurs. Au fil du temps, des groupes de travail se sont constitués dont les objectifs étaient d'enrichir la spécification et l'ajout de nouvelles fonctionnalités dont les objets métiers, les interfaces de domaines, etc. (voir Figure III.2, [OMG 1997a, OMG 1998, Soley et Stone 1995]).

[12] Adaptation du terme anglophone "runtime".

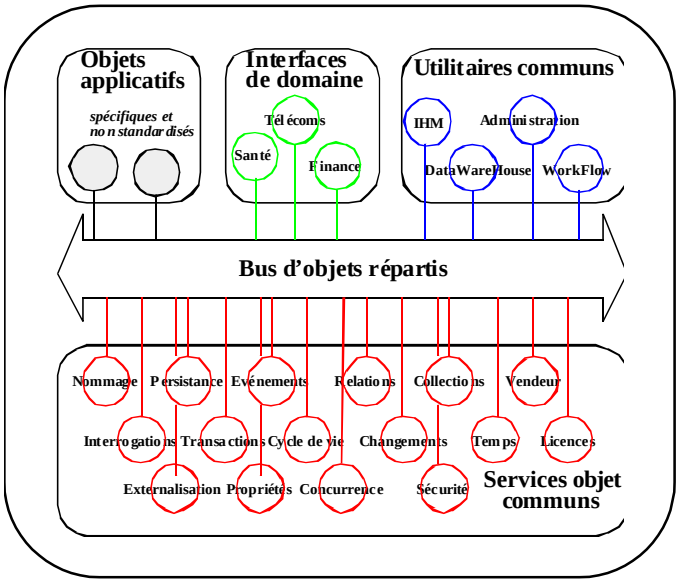

Figure III.2 Architecture de l'environnement CORBA (ou OMA – Object Management Architecture)

Les principales limitations de la spécification CORBA 2 sont les suivantes [Wang et al. 2000] :

- aucune proposition standard pour le déploiement des implémentations des objets ;
- limitation des possibilités d'extension des fonctionnalités des objets ;
- la disponibilité des services objet CORBA n'est pas définie à l'avance ;
- pas de gestion standard du cycle de vie des objets.

A la suite de plusieurs remarques sur l'absence de spécifications sur les composants d'une part, et l'émergence de propositions concurrentes pour la spécification de composants (EJB, COM/DCOM, .NET) d'autre part, l'OMG (*Object Management Group*) décida de définir une spécification des composants qui viendrait enrichir la spécification CORBA 3 : c'est le modèle de composant CORBA ("CORBA Component Model" – CCM). Ce modèle prend en compte les aspects diffusion et déploiement d'applications. Nous notons que la proposition sur les composants CORBA s'est fortement inspirée du modèle EJB de Sun (notons d'ailleurs que les deux propositions sont entièrement compatibles).

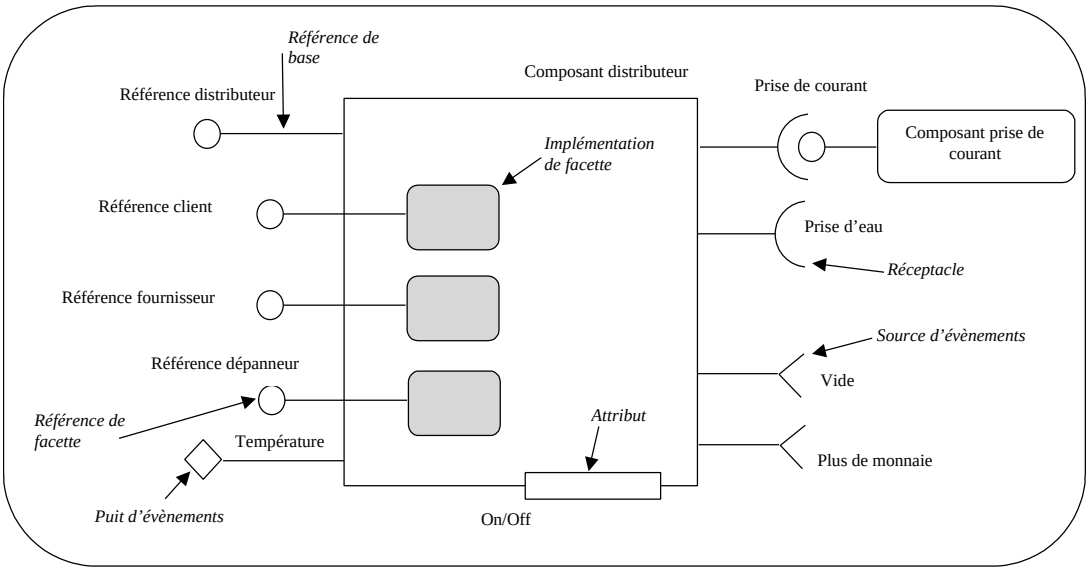

Figure III.3 Modèle abstrait de composant CORBA [Marvie 1999]

A l'image des autres propositions, l'objectif principal est de fournir aux développeurs de composants une spécification à partir de laquelle des composants pourraient être développés mais également, inter-opérer via le bus logiciel CORBA.

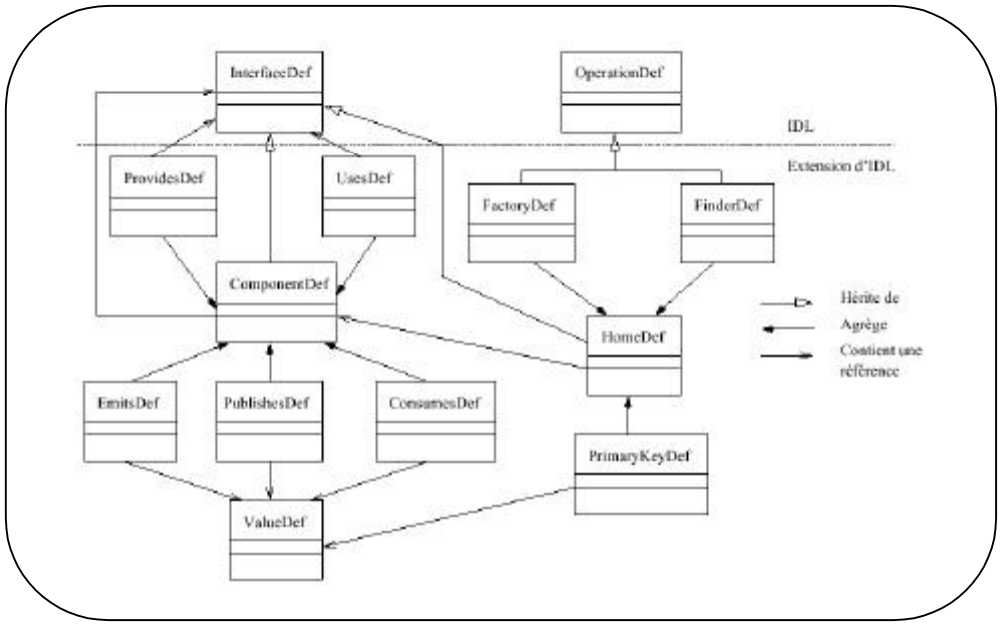

Figure III.4 Méta-modèle de composants suivant une extension IDL [Marvie 1999]

L'enrichissement de la spécification CORBA porte sur un nouveau méta-type [Marvie et al. 2000] : le composant (component) auquel sont rattachés plusieurs propriétés permettant de définir le composant, les services qu'il fournit, ses attributs, les évènements auxquels il souscrit, les évènements qu'il publie, etc. L'un des intérêts majeur est évidemment le couplage avec le bus logiciel permettant de se reposer sur les services CORBA (gestion des transactions, persistance, sécurité, etc.) ; mais nous noterons aussi la mise à disposition d'un cadre permettant de définir des composants "serveurs". Le méta-modèle de composant est basé sur UML et dispose de projection vers le MOF (Meta Object Facility).

Les atouts majeurs de cette approche sont (1) qu'elle émane d'un organisme reconnu, (2) qu'elle est indépendante d'un langage de programmation particulier (elle se base sur l'OMG-IDL contrairement à l'approche EJB qui utilise le langage Java, etc.), (3) qu'elle permet d'encapsuler les autres approches (EJB, DCOM, etc.), (4) qu'elle fournit un environnement de déploiement.

III.2.1.7 Evaluation

En fait, les différentes approches permettent de développer des composants *a priori*, respectant telle ou telle spécification. Par contre, elles s'adressent au développement de composants de granularité assez fine et ne prennent pas en compte les composants gros grain comme le sont les produits COTS, selon une approche boîte noire.

En outre, les composants ainsi développés ne peuvent être utilisés que dans le cadre de l'environnement dont ils respectent la spécification, ce qui ne leur permet pas d'être réutilisés ailleurs.

Les diverses propositions sont fortement liées à une technologie permettant le développement d'applications réparties que ce soit, COM/DCOM utilisant DCE, CCM utilisant CORBA, EJB avec Java/RMI en toile de fond. D'ailleurs, la définition de composant proposée dans

[Bachman et al. 2000] fait l'hypothèse qu'il existe toujours un modèle de composant spécifique [D'Souza et Wills 1999, Wills 1999] sous-jacent.

Nous remarquons également que la plupart de ces approches se sont initialement concentrées sur les propriétés que devaient respecter les composants. Petit à petit, les versions successives des spécifications ont tendance à proposer des modèles d'implémentation, des modèles de développement et des modèle de déploiement (d'une façon générale, s'intéressent aux différentes phase du cycle de vie du composant). Cependant, ces tendances sont très récentes.

Nous dégageons de cette étude les points suivants :

- consensus sur un modèle générique de composant ;
- caractérisation du composant comme "pièce importante" du logiciel et des applications ;
- réutilisation des composants.

En revanche, nous notons les limitations de ces approches :

- spécifications récentes et souvent incomplètes dans le sens où elles n'intègrent pas toutes les catégories de composants (vision parcellaire) ;
- visions "propriétaires" (fortement liées à des technologies pour les applications distribuées) imposant un certain nombre de restrictions et de contraintes ;
- propositions parfois complexes, dont l'objectif est l'exhaustivité qui requière un haut degré d'expertise ;
- existence d'un modèle de composant imposé.

Les propriétés des systèmes à base de composants

Dans les systèmes à base de composants, les propriétés suivantes ont été identifiées :

- les aspects liés à la gestion des transactions ;
- la gestion de la persistance ;
- la gestion des accès aux ressources partagées.

Les approches que nous avons présentées prennent en compte la gestion de ces propriétés. En particulier, les approches EJB, bus logiciels respectant les spécifications CORBA, les objets répartis DCOM proposent des technologies permettant de gérer les transactions (JTA et JTS pour Java, service de gestion des transactions des bus logiciels CORBA, MTS – Microsoft Transaction System pour les objets DCOM, etc.). La gestion de la persistance est essentiellement assurée dans les approches de type EJB.

III.2.2 Les architectures logicielles

Développer des applications logicielles implique de nombreuses ressources et compétences. Au fur et à mesure que les besoins augmentent, les applications se complexifient. Parallèlement, l'intégration de composants existants (composants issus du marché, applications patrimoines) dans un système existant ou futur devient un des défis majeurs que les concepteurs de systèmes doivent relever (les contraintes de coûts, de délais, de qualité, etc.). Les travaux portant sur les architectures logicielles ont pour but principal de fournir un niveau d'abstraction permettant de spécifier la composition de systèmes et des éléments logiciels de ces systèmes [Perry et Wolf 1992, Shaw et Garlan 1996]. Cependant, pour pouvoir décrire les architectures logicielles permettant d'aider les acteurs du développement

de logiciels, il est nécessaire de fournir des notations, des méthodes, des outils, supports à la description des architectures logicielles de telle sorte qu'il est possible d'en étudier les propriétés, de faire des vérifications et des analyses.

III.2.2.1 Les entités conceptuelles

Les bases des architectures logicielles reposent sur l'abstraction des éléments d'architecture, de leur composition et de la définition de styles architecturaux qui vont guider le processus de développement [DSoS 2000]. Les éléments suivants font l'objet de consensus dans la communauté des architectures logicielles :

> ***composant*** qui est une abstraction caractérisant une unité de calcul ou de stockage de données. En règle générale, la spécification d'un composant décrit le comportement du composant ainsi que ses points d'interface (les interfaces fournies et requises qui sont souvent référencées comme les ***ports*** ou les ***rôles***) avec les autres éléments de l'architecture ;
>
> ***connecteur*** qui est une abstraction caractérisant un modèle de composition entre composants. Un connecteur définit le protocole d'interaction entre composants et au travers duquel les composants sont composés. En outre, une spécification du connecteur fournit une spécification du comportement du connecteur ainsi que ses points d'interface (souvent référencés comme ***ports*** ou ***rôles***) avec les autres éléments de l'architecture ;
>
> ***configuration*** qui définit la structure de (sous-)systèmes en composant des collections d'instances de composants au travers de correspondance (ou branchements) définis explicitement par les instances de connecteurs. L'architecture logicielle d'un système est alors définit comme une configuration avec ses types de composants et ses types de connecteurs instanciés pour cette configuration donnée. Il est possible qu'une configuration soit intégrée à l'intérieur d'un système ; dans ce cas, elle correspond soit à un composant, soit à un connecteur pour lequel l'architecture a été définie.

Nous pouvons remarquer (et constater) que la distinction entre composant et connecteur est source d'erreur et d'incompréhension...Car le concept de connecteur n'est pas encore bien défini. En particulier, certains travaux du domaine montrent qu'il est possible de ne considérer que des composants et de définir des configurations en ne spécifiant que les branchements directs entre composants (voir [Magee et al. 1995]). Cependant, il semble que le concept de connecteur soit d'un intérêt majeur lorsque l'on s'intéresse au développement de systèmes distribués complexes [DSoS 2000], notamment en permettant de décrire des fonctions de gestion du système distribué encapsulées dans des infrastructures de type bus logiciel. [Allen et Garlan 1997] ont montré que la spécification de connecteurs permettait d'appliquer (et de faire appliquer) des modèles d'interaction entre composants.

Cependant, l'ensemble des éléments présentés précédemment ne suffisent pas à l'élaboration d'architectures logicielles de systèmes. La notion de ***style d'architecture*** permet de définir des caractéristiques communes entre systèmes qui faciliteront leur analyse et leur développement. Un style d'architecture définit un ensemble de propriétés qui sont partagées par les configurations appartenant (membres) à ce style. Ces propriétés peuvent, par exemple, définir le type de connecteurs à utiliser (lorsque l'infrastructure sous-jacente est partiellement figée), définir des contraintes sur la topologie du système (par exemple, les connecteurs ne peuvent pas être directement connectés), etc.

III.2.2.2 Les formalismes

De nombreux travaux se sont intéressés à fournir des moyens permettant de modéliser les systèmes et leurs architectures. Les propositions dans ce domaine se divisent

schématiquement en deux catégories selon qu'elles s'intéressent à la définition de notations pour la description des architectures ou au processus de développement à base d'architectures (voir Figure III.5).

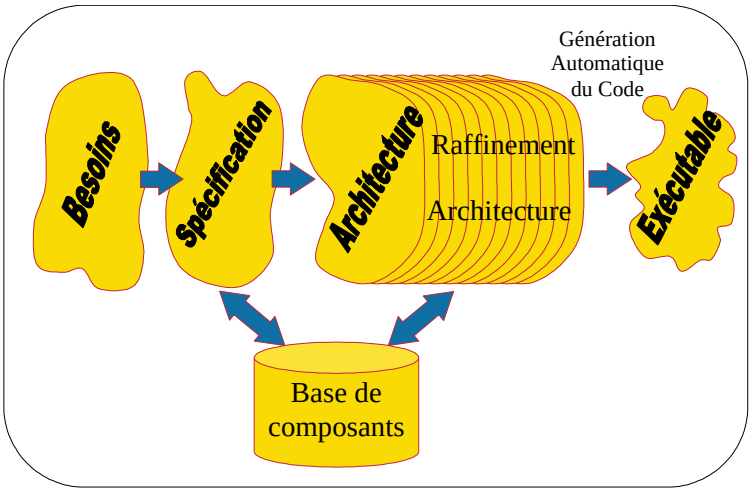

Figure III.5 Processus de développement selon une vision "architecture logicielle"

La première catégorie concerne la définition de *Langages de Description d'Architectures* (ADL)[13] largement promue par le monde académique alors que la seconde catégorie est davantage le résultats d'efforts industriels comme le Processus Unifié de Rational (RUP [Kruchten 1999]). Nous noterons que, plutôt que de définir des notations nouvelles pour la description de l'architecture, ces processus reposent sur la notation standard UML (*Unified Modelling Language*) [OMG 1997b, OMG 1997c]. Nous remarquons toutefois que le formalisme UML n'est pas totalement accepté comme ADL par la communauté des architectures logicielles étant, par nature, porté sur la conception orientée-objet et donc, ayant intrinsèquement un faible niveau d'abstraction, contrairement aux ADL. D'autre part, ce formalisme n'est pas réellement adapté à la vérification de propriétés et l'analyse des systèmes, souffrant d'un manque de rigueur. Cette restriction fait l'objet de travaux qui visent à associer UML avec le langage OCL (*Object Constraint Language*) [OMG 1997d] ce qui permettrait de spécifier des contraintes sémantiques sur des modèles de systèmes basés sur UML. Ces contraintes seraient décrites en termes de prédicats d'une logique de premier ordre.

En ce qui concerne les langages de description d'architectures, ils sont plus nombreux et s'intéressent à différents aspects liés au développement et aux architectures des systèmes. La Figure III.6 propose une taxonomie des approches prises.

[13] Nous garderons l'abréviation anglo-saxonne ADL (Architecture Description Language).

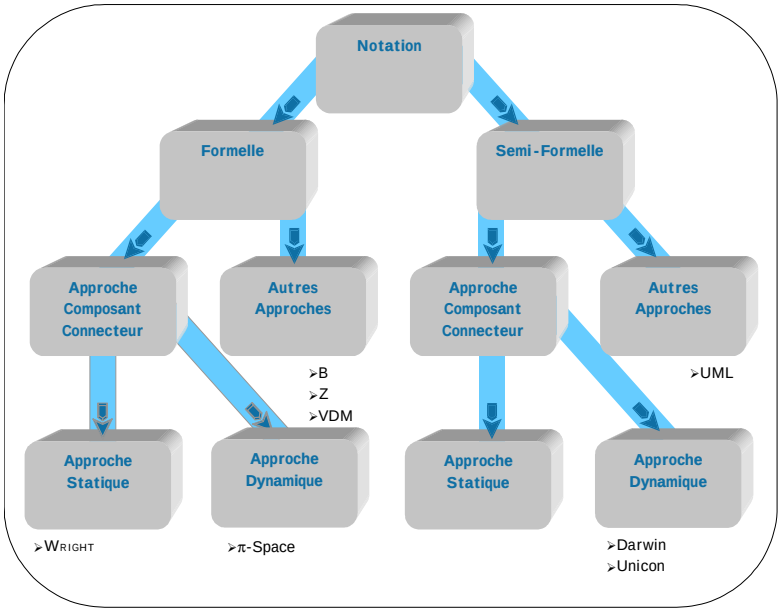

Figure III.6 Taxonomie des approches dans le domaine des architectures logicielles

Grossièrement, les langages de description d'architecture s'intéressent aux aspects suivants :

conception de système. C'est l'objectif principal des ADL. Cependant, les ADL diffèrent dans l'assistance qu'ils procurent au raffinement d'architecture. En général, l'approche reste classique, reposant sur la définition d'éléments architecturaux et permettant l'étude du processus de raffinement [Medvidovic et al. 1999, Bolusset et al. 1999] ;

l'analyse de système. Certains ADL se sont concentrés sur l'analyse comportementale des systèmes en fournissant également des technologies permettant l'analyse et la vérification de modèles. Par exemple, le langage Wright [Allen et Garlan 1997] appartient à cette catégorie de ADL ; il est basé sur CSP et est couplé avec l'outil FDR pour vérifier les possibles libérations d'étreintes fatales. L'utilisation de CSP permet de faire des vérifications de propriétés concernant l'intégrité du système, notamment en vérifiant que les connecteurs sont utilisés conformément au protocole d'interaction entre les composants. Cependant, l'utilisation de CSP ne permet pas la description d'architectures dynamiques [Allen et al. 1997] contrairement au langage π–space [Chaudet et al. 2000a, Chaudet et al. 2000b] basé sur le π–calcul ;

construction de système. Le dernier aspect concerne les ADL qui assistent les acteurs du processus de développement de logiciels et particulièrement dans la construction de ces derniers. Dans ce contexte, le langage est associé à des outils qui permettent de générer des configurations exécutables à partir de descriptions d'architectures. Parmi ces langages nous trouvons C2 [Medvidovic et al. 1999], Darwin [Magee et al. 1997] et Unicon [Shaw et al. 1995].

III.2.2.3 Evaluation

Nous pouvons trouver dans la littérature, des analyses et des comparaisons des travaux émergeants ces dernières années dans le domaine des architectures logicielles [Medvidovic et Rosenblum 1997, Medvidovic et Taylor 2000, DSoS 2000]. A partir de ces articles et de ce que nous avons présenté dans ce document, nous pouvons établir les avantages et les inconvénients des travaux portant sur les architectures logicielles.

Les avantages :

- bonne spécification des systèmes reposant sur des formalismes formels ou semi-formels et permettant la vérification de propriétés ;
- haut niveau d'abstraction dans la définition des éléments architecturaux ;
- bons supports à la spécification de l'architecture. Certaines approches permettent également de générer du code par raffinements successifs.

Cependant, leurs limitations concernent principalement :

- la couverture du processus de développement : chaque ADL se concentre sur une ou plusieurs phases du processus de développement. Peu de langages sont exécutables directement (il faut recourir à un enchaînement de formalismes, d'outils associés et d'abstractions pour, en partant de la description formelle, arriver au code exécutable) ;
- l'évolution de l'architecture : cette évolution, quant elle est permise, repose essentiellement sur des langages conceptuels de haut niveau [Chaudet et al. 2000b] mais qui ne permettent pas de se soustraire des phases avales du processus de développement ;
- hétérogénéité des approches et des formalismes ;
- les hypothèses faites concernant le type de composants : les langages proposés permettent une description des architectures et des systèmes *a priori* (l'approche est déductive) alors que notre problématique concerne également une approche où les composants existent par ailleurs, indépendamment du contexte d'utilisation mais dont le comportement ne peut être (totalement ou partiellement) modélisé ;
- l'abstraction : le niveau d'abstraction utilisé pour décrire des architectures n'est pas toujours en correspondance avec les composants "réels" (c'est-à-dire des entités logicielles autonomes) ;
- le processus de raffinement s'accompagne généralement de pertes d'informations (spécifications et propriétés). Des travaux actuels portent sur le raffinement [Bolusset et al. 2000] ;
- non prise en compte du processus sujet (le but de l'application résultante) et la faible description des données manipulées (seules les entités de calcul sont décrites ; les entités qui manipulent les données sont bien souvent ignorées) ;
- non prise en compte des technologies de distribution en ce qui concerne les systèmes distribués et recouvrements avec les langages de description d'interface (IDL).

En fait, les travaux portant sur les architectures logicielles s'intéressent essentiellement à la modélisation, à l'analyse et à la vérification de propriétés des architectures logicielles de systèmes. Le niveau d'abstraction étant souvent élevé, l'étude de l'exécution des systèmes ainsi modélisés (étudiés) n'est que peu abordée. D'autres travaux de recherche ont eu pour objectif de fournir des langages à partir d'approches multi-agents, de paradigmes différents et suivant des objectifs différents, des techniques permettant l'assemblage d'outils logiciels. Ce sont ces approches, ces techniques que nous allons étudier dans la suite de ce chapitre (voir section III.4.2).

III.2.3 Des composants aux COTS

Des travaux relativement récents ont débuté au sein de l'institut du génie logiciel (Software Engineering Institute - SEI) de Carnegie Mellon et portant sur l'étude des systèmes à base de

composants issus du marché (appelés dans la littérature les CBS pour "COTS-Based System").

Les études effectuées par le SEI portent essentiellement sur l'intégration de COTS au sein de système en termes de coûts et de difficultés, d'apprentissage, de bouleversements, etc. En fait, leur approche principale part de la constatation suivante : les utilisateurs expriment des besoins que les systèmes doivent satisfaire ; or, si on considère que ces systèmes vont justement être à base de composants, il faut être en mesure de "qualifier" les constituants (composants) des systèmes afin de savoir s'ils vont nous permettre de répondre aux besoins.

D'après les travaux effectués au sein du SEI, tout système basé sur des composants issus du marché devrait respecter le cycle de vie présenté par la Figure III.7.

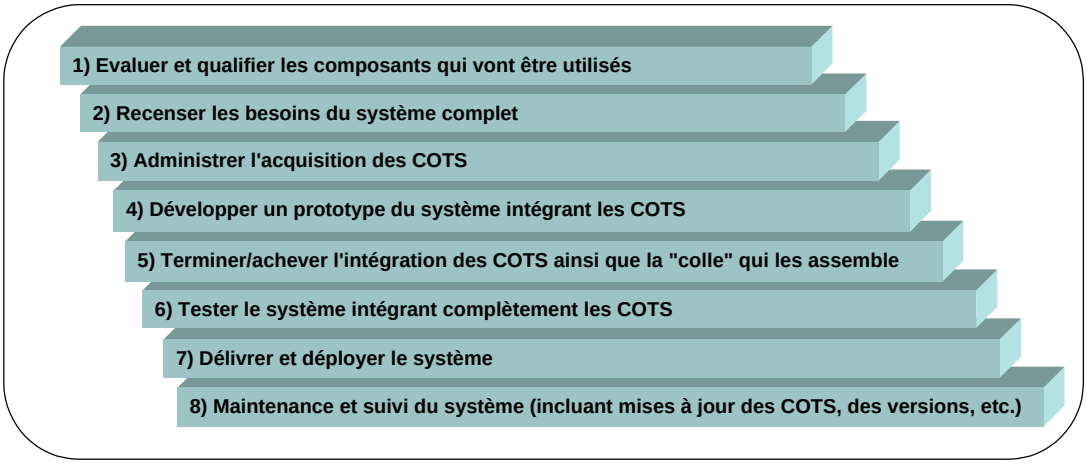

Figure III.7 Le cycle de vie d'une application utilisant des composants issus du marché [SEI 2001]

Nous allons, dans la suite de ce document, regarder les différentes étapes de ce cycle de vie qui permettent de mieux comprendre le développement de systèmes à base de composants.

III.2.3.1 Définition des composants issus du marché (COTS)

Les travaux effectués par le SEI ont permis de proposer une définition des composants issus du marché (COTS).

> Un composant issu du marché est :
>
> - un produit logiciel commercial,
> - dont le code source est indisponible[14],
> - ayant des mises à jour périodiques s'accompagnant d'un accroissement des fonctionnalités fournies alors que certaines autres deviennent obsolètes.

Ainsi, de cette définition, nous pouvons dire :

> - qu'aucun contrôle sur les fonctionnalités ou la performance du produit logiciel peut être assurée ;
> - que les composants de type COTS ne sont pas conçus pour interopérer avec d'autres ;
> - qu'aucun contrôle ne peut être assuré sur l'évolution du composant ;

[14] Des études faites au sein du SEI ont montré qu'au moins 30% des composants de type COTS qui sont utilisés doivent avoir leur code source modifié pour être intégrés.

- que les comportements des distributeurs/éditeurs de composants de type COTS divergent largement (ils sont donc imprévisibles).

Cette définition est reprise et détaillée dans [FAR 1996].

III.2.3.2 Un nouveau processus de développement ?

En fait, l'approche prise par [Brown et Wallnau 1996] est de caractériser les différentes activités essentielles au développement de systèmes basés sur des composants[15] issus du marché ; il s'agit de la description d'un processus tel qu'il est montré par la figure suivante.

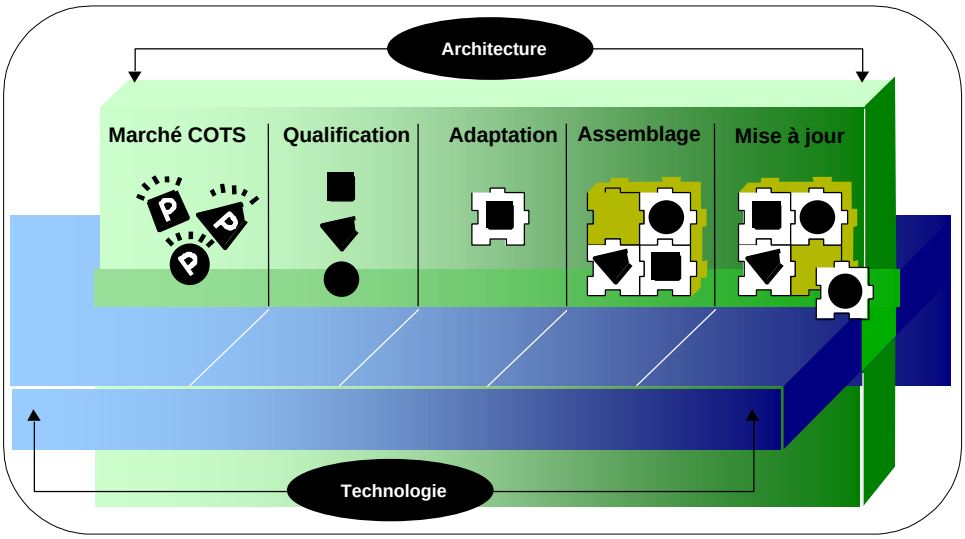

Figure III.8 Les activités du processus de développement de logiciel à base de composants (d'après [Brown et Wallnau 1996, SEI 1997])

On trouve les activités principales du processus :

1. l'évaluation et la qualification des composants ;
2. l'adaptation des composants ;
3. l'assemblage des composants pour former un système ;
4. l'évolution du système.

III.2.4 Le développement de systèmes à base de COTS

III.2.4.1 L'évaluation et la qualification des composants

La première activité concerne principalement :

- l'identification et la caractérisation des composants face à l'expression des besoins ;
- l'évaluation des différentes technologies qui sont offertes par le marché ;
- l'impact d'une nouvelle version d'un composant sur un système installé ;
- l'impact du déploiement d'un composant issu du marché.

[15] On trouve dans la littérature les abréviations anglophones CBSD pour "Component-based software development" (développement de logiciel à base de composant) et CBSE pour "Component-based software engineering" (génie logiciel à base de composant).

Ces objectifs doivent prendre en compte plusieurs critères (pour l'évaluation et l'opportunité d'utiliser tel ou tel composant issu du marché) tels que [Abts 1997, Anderson 1989, Beach et Bonewell 1993] (1) l'éditeur/le vendeur du composant, (2) la fréquence des versions et mises à jour, (3) les technologies supports et le respect (ou non) de standards, (4) le coût d'une intégration potentielle (intégrant moyens humains et compétences,...), (5) les bénéfices que tirera le système qui intégrera le composant issu du marché, etc. En fait, l'ensemble des études montrent que l'évaluation et la sélection des composants issus du marché est un axe de recherche particulièrement important, préliminaire à l'intégration des composants [Jung et al. 1999]. Plusieurs études portent sur le choix de critères, sur la mesure des composants issus du marché à partir de ces critères puis sur l'agrégation des mesures en vue de tirer une évaluation complète du composant [Abts et Boehm 1997].

Le cas spécifique des composants de type COTS est l'opportunité de les utiliser pour une application. Cette opportunité est à l'intersection de trois considérations comme le montre la figure suivante.

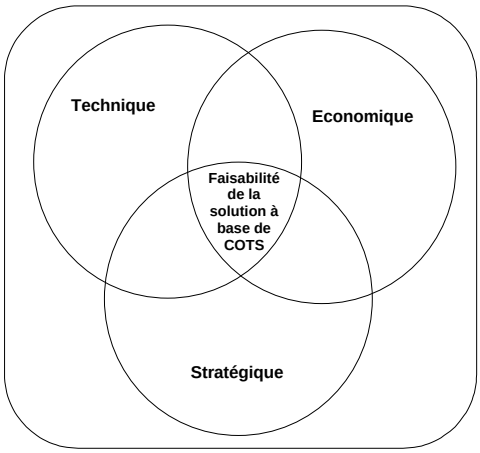

Figure III.9 Les contraintes techniques, économiques, stratégiques influentes sur la solution envisagée (d'après [SEI 2001])

III.2.4.2 L'adaptation...souvent nécessaire

La seconde activité porte sur la nécessaire adaptation des composants en vue de leur utilisation dans un "contexte". L'idée est que tout composant permet de satisfaire (généralement) plusieurs besoins en se basant sur des hypothèses liées fortement aux contextes dans lesquels le composant va être utilisé. Le degré d'accessibilité de la structure interne du composant suggère différentes approches pour l'adaptation de ce dernier [Valetto et Kaiser 1995] :

- *"boîte blanche"* où l'accès au code source permet de modifier le composant de manière à ce qu'il puisse coopérer avec les autres composants ;
- *"boîte grise"* où le code source n'est pas disponible mais le composant fournit son propre langage d'extension ou une interface programmable (API) à partir desquels il est possible d'influer sur son comportement ;
- *"boîte noire"* où ni le code source ni un langage d'extension ou une interface programmable ne sont disponibles.

On peut cependant ajouter que l'approche de type "boîte blanche" présente l'inconvénient majeur de devoir maintenir les changements effectués dans le code source du composant : (1) les risques de déviation entre le composant d'origine et le composant modifié sont importants d'une part et (2) les changements devront pouvoir être intégrés à toute nouvelle version du

composant d'origine. Ces deux points concernent principalement la pérennité des modifications introduites.

Les deux autres approches ("boîte grise" et "boîte noire") sont, de nos jours, les plus fréquemment rencontrées ; nous pouvons effectivement dire que les services de base sont relativement bien couverts par les applications actuelles et que les nouveaux besoins peuvent être satisfaits en composant ces services (la façon de composer définit le caractère original de l'application résultante). Les approches "boîte grise" et "boîte noire" présentent, contrairement à l'approche "boîte blanche", l'inconvénient majeur de ne pas rendre accessible (au moins partiellement) le code interne du composant.

III.2.4.3 L'activité d'assemblage de composants

Nous trouvons, dans la littérature, différents termes pour qualifier l'assemblage de composants en un système, parmi lesquels :

- *assemblage* [Brown et Wallnau 1996, SEI 1997],
- *composition* [Bachman et al. 2000],
- *intégration* [Wasserman 1989].

Le terme de "colle" a été introduit pour qualifier l'assemblage de parties logicielles (analogie avec le moyen permettant l'assemblage de matériaux – voir Figure III.10) :

> développement logiciel in-situ[16] et composé de (1) la partie programmation (code) permettant les échanges d'information entre le composant (issu du commerce ou système patrimoine) et le système ou d'autres composants avec lesquels il doit s'intégrer, (2) la partie programmation permettant de connecter (d' "arrimer") le composant au système ou aux autres composants mais qui ne permet pas l'échange d'information, (3) la partie programmation qui va fournir ou rendre disponibles (exhiber) des services absents au niveau du composant mais dont ce dernier est tributaire.

Si les deux conditions font l'objet d'un large consensus dans la littérature, le troisième point est davantage controversé [SEI 2001] ; le fait est qu'un composant issu du marché est censé fournir une liste de services qui ne sont pas forcément tous implémentés. Cet inconvénient majeur rentre en compte dans la sélection du composant (sa possible remise en cause d'ailleurs), mais n'intervient pas dans l'objet de notre étude. Nous ne reviendrons pas sur ce point dans la suite du document.

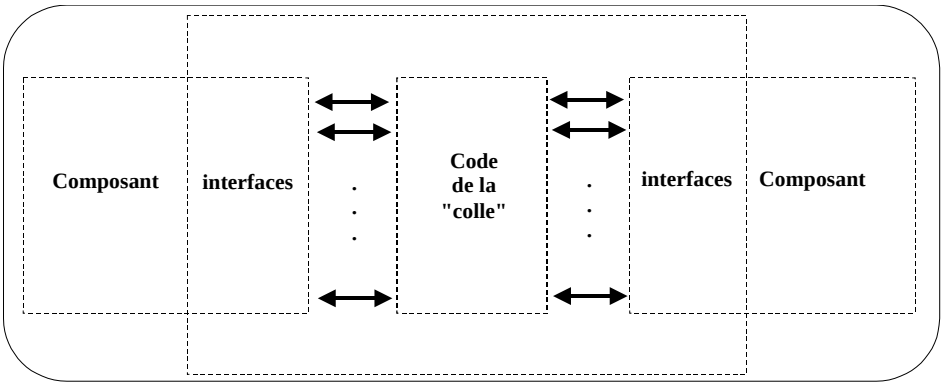

Figure III.10 La "colle": échanges d'informations, connexions des interfaces, services requis (d'après [Brown et Wallnau 1996])

[16] Partie de code qui ne peut être envisagée en dehors du contexte d'intégration (réalisée au moment de cette intégration et sur le "lieu" même de l'intégration).

L'idée générale est de faire un tout à partir de parties. Ces différentes terminologies traduisent en fait des nuances dans la façon d'assembler les composants (en particulier, l'*intégration* a souvent un sens fort, à savoir qu'un composant intégré perd son identité propre).

Le terme *interopérabilité*, que nous allons présenter plus loin dans ce chapitre est intrinsèquement plus souple dans la mesure ou, contrairement à l'intégration, les composants sont censés préserver leur identité.

Dans le domaine du CBSE, les techniques d'assemblage sont "standardisées" dans le sens où chacune des propositions (EJB, DCOM, CCM-CORBA, etc.) repose sur une technologie particulière permettant la construction d'applications (Java et Java/RMI, IDL et bus CORBA, environnement pour les EJB, etc.).

Les travaux traitant des composants issus du marché dans le domaine de l'ingénierie des systèmes à base de composants (CBSE) ont consacré très peu de travaux sur les techniques d'assemblage et se sont surtout inspirés de techniques issus d'autres domaines (que nous aborderons plus loin dans ce chapitre – voir section III.4.2).

III.2.4.4 L'évolution des systèmes à base de COTS

La prise en compte du phénomène de composition des systèmes permet d'affiner l'étude de l'évolution de ces derniers. Tout particulièrement, la prise en compte de composants issus du marché à un impact important sur le processus d'évolution.

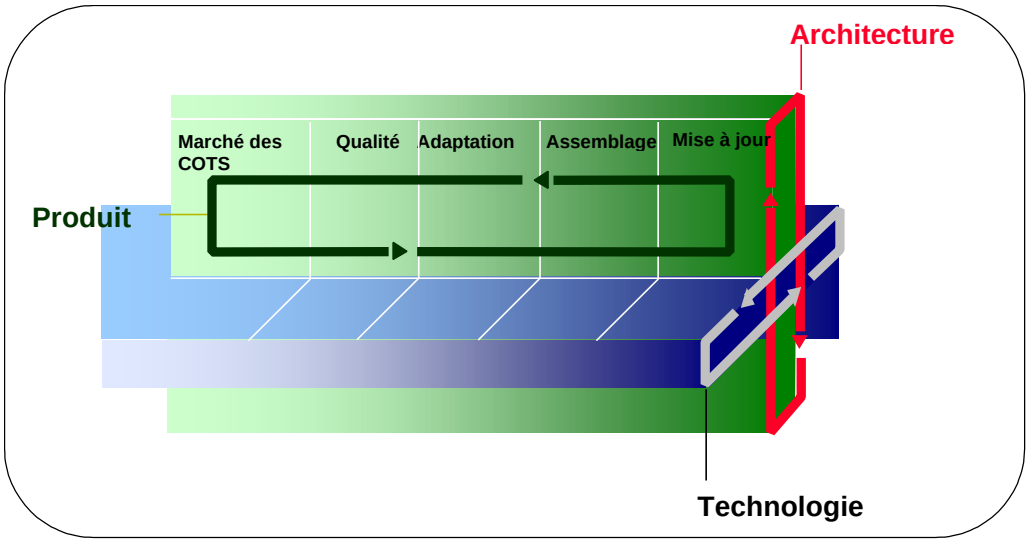

Figure III.11 L'évolution en trois dimensions (d'après [SEI 1997])

Comme le montre la Figure III.11, l'évolution de système à base de composants issus du marché est tridimensionnelle :

1. l'évolution des différents composants (produits logiciels), constituant l'environnement, y compris les mises à jour des produits, etc. ;
2. l'évolution des technologies qui entrent en jeu dans chacune des phases du processus de développement de systèmes à base de composants issus du marché ;
3. l'évolution de l'architecture du système (ajouts/retraits de composants, évolution du mode de fonctionnement, de la politique).

Les facteurs de ces évolutions sont pour la plupart bien connus parmi lesquels nous pouvons citer l'expressions de nouveaux besoins, l'obsolescence des supports existants, etc. D'autres critères rentrent en jeu (coûts, performance, compétences humaines, etc.) dans le processus

d'évolution de tout système. L'étude de l'évolution seule fait l'objet de quelques travaux [Bergey et al. 1997].

En fait, les différentes facettes qui ont une large influence sur l'évolution d'un système sont les suivantes [Bergey et al. 1997] :

- le projet qui est à mener ;
- l'organisation de l'entreprise qui va devoir mener le projet ;
- les technologies à disposition et qui servent de support :
    - à l'ingénierie du logiciel ;
    - à l'ingénierie des systèmes ;
- les systèmes existants.

L'ensemble de ces facettes va permettre de définir une stratégie d'évolution du système en place vers un système "adapté"[17].

Puisque notre approche est essentiellement portée sur les phases d'adaptation et d'assemblage des composants, l'évolution des systèmes à base de composants selon cette restriction va concerner : (1) l'évolution de la composition du système (ajout/retrait de composants), (2) l'évolution de l'assemblage des composants entre eux (de la "colle").

Dans la prise en compte de l'évolution telle qu'elle vient d'être présentée, nous ne prenons pas en considération les aspects liés à l'évolution intrinsèque du processus qui est, d'une certaine manière un autre sujet et qui a été traité par ailleurs (voir chapitre 2, [Cunin et al. 1999, Cunin 2000, Cîmpan 2000, PIE 1999, Promoter 1999, Avrilionis et al. 1996]).

L'impact des technologies

Les technologies de l'information et tout particulièrement l'avènement de technologies permettant la distribution de système, d'applications (CORBA [OMG 1998], Java/RMI [Sun 1997], DCOM [DCOM 1997]) mais aussi de nouvelles méthodes de conception (orientée-objet par exemple) ont une influence considérable sur l'évolution des systèmes au cours du temps (on parle alors de ré-ingénierie).

Il est clair que ces technologies ont permis principalement deux choses importantes :

- repousser la durée de vie d'une application existante en étendant ses fonctionnalités,
- l'assemblage de différentes applications, composants et autres pour en former de nouvelles.

L'inévitable évolution

Ainsi, tout système à base de composants est au centre de sollicitations qui peuvent avoir éventuellement des actions neutralisantes mais dont le résultat est une inévitable évolution. Les systèmes sont tirés vers l'évolution par les contraintes technologiques et sont poussés vers cette même évolution par l'apparition de nouveaux besoins.

III.2.4.5 Evaluation

Les travaux étudiés dans cette partie concernent essentiellement des composants de gros grains, composants particulièrement autonomes et pour certains, des composants existants du marché. Selon l'approche prise par [Fox et al. 1997] qui présente un processus de

[17] L'adjectif qualifie le fait que le système est au centre des facettes et fait donc l'objet d'un consensus.

développement pour les systèmes à base de produits de type COTS, notre approche se situe dans la phase intitulée "phase de construction". Notre objectif étant de fournir un cadre architectural permettant d'intégrer les outils de type COTS et de permettre l'évolution de l'infrastructure.

Selon cet objectif, l'ensemble des travaux présentés dans cette section permettent de mettre en évidence certaines avancées :

- prise en compte et caractérisation des composants issus du marché ;
- adaptation du modèle "classique" du développement de logiciels en tenant compte de la spécificité des composants issus du marché.

Par contre :

- les travaux portant sur les composants issus du marché se sont consacrés pour l'essentiel, aux premières phase du processus de développement; les phases d'adaptation et d'assemblage faisant l'objet d'approches pragmatiques (pas de méthodologie ni de modélisation) ;
- les propriétés que nous avons énumérées (gestion des transactions, persistance, etc.) sont certes recensées mais n'ont fait l'objet d'aucune proposition à notre connaissance. Ceci s'explique essentiellement par le fait que la gestion de ces propriétés imposent des contraintes fortes et hypothèses sur les composants qui vont à l'encontre de la caractéristique principale des COTS (non disponibilité du code et non prise en compte des propriétés sus-mentionnées dans la majorité des composants issus du marché).

III.3 L'intégration d'applications d'entreprise (EAI)

Un domaine qui a actuellement le vent en poupe est celui de l'intégration d'applications d'entreprises[18] dont l'objectif se rapproche considérablement de ceux déclinés dans cette thèse, à savoir, fournir aux utilisateurs un environnement résultant de l'intégration d'applications logicielles existantes (issues du marché ou applications patrimoines).

III.3.1 Les objectifs

Les objectifs de ce domaine visent à "lier" de façon optimale divers systèmes et applications de l'entreprise afin [Schmidt 2000] :

1. de constituer un support efficace aux processus de cette dernière (processus manufacturiers, processus économiques, etc.),
2. de lui permettre de répondre aux changements du marché.

Il s'agit, dans la démarche, de motivations commerciales et économiques qui vont influencer des décisions stratégiques dont l'aboutissement se traduit par l'intégration d'applications existantes aux seins des entreprises. Les clients, les fournisseurs d'une entreprise sont très largement impliqués. En particulier, les chaînes logistiques (*supply-chain*) sont un terrain favorable pour l'EAI parce que les entreprises communiquent, échangent de l'information, etc.

Ainsi, intégrer des applications ne peut se considérer sans prendre en compte la notion de "retours sur investissements" et de "bénéfices" qu'ils soient directs ou indirects.

[18] Le terme anglophone est Enterprise Application Integration (EAI) dont l'abréviation sera utilisée dans ce document.

Dans ce contexte, l'EAI représente "un ensemble d'outils, de progiciels d'intégration qui offrent des moyens de connecter et de faire communiquer entre elles les applications de l'entreprises, existantes ou à venir" [Hostachy 2000].

Les raisons principales d'une telle intégration sont les suivantes [Stonebraker 1999] :

- présenter à l'utilisateur final une vision unifiée de l'information gérée par les différentes applications afin de l'aider dans ses prises de décision ;
- résoudre le problème de la cohérence entre des systèmes qui s'entrecroisent.

III.3.2 Un modèle de "maturité" pour l'intégration d'applications

Tout récemment, un organisme américain (*American Management Systems*) a proposé un modèle permettant de valider le degré d'intégration des applications des entreprises (voir Figure III.12). Ce modèle a pour but de guider les entreprises dans les étapes conduisant à la gestion optimale de leurs capacités d'intégration d'applications en vue de satisfaire les objectifs de l'entreprise (économiques, technologiques, etc.).

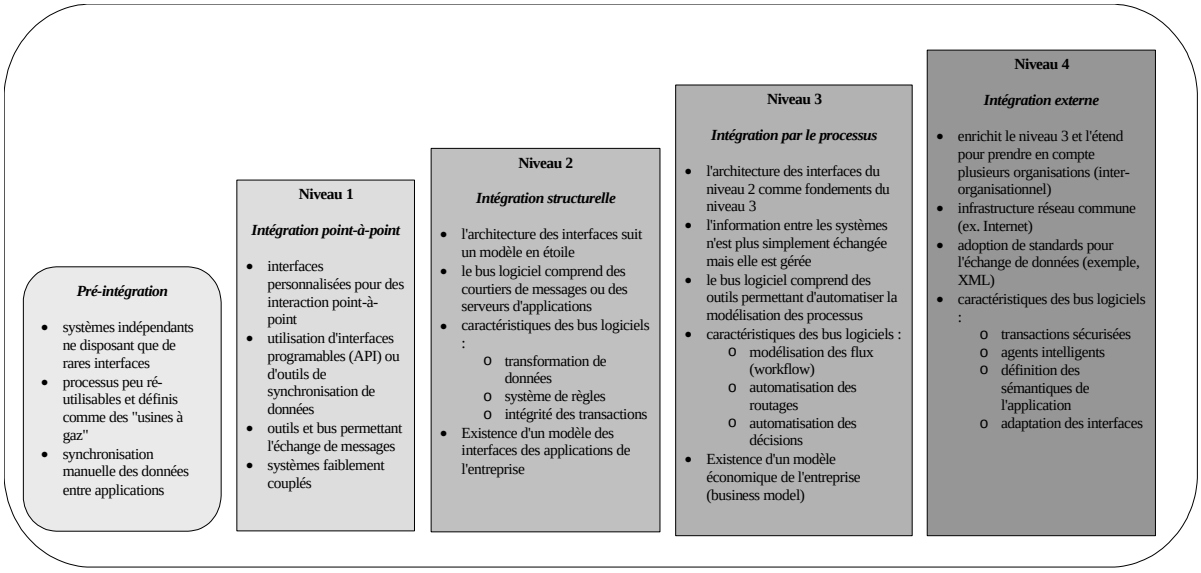

Figure III.12 Modèle de maturité - Intégration d'applications d'après [Schmidt 2000]

Les entreprises se confrontent à ce schéma pour améliorer leurs systèmes et l'intégration des applications avec les partenaires. Pour ce faire, elles bénéficient de technologies étant de plus en plus adaptées :

- les bus logiciels intégrant de nombreux services (gestion des transactions, déploiement, etc.),
- des applications issues du marché respectant des "standards" et qui induisent de faibles changements au niveau des systèmes.

Les objectifs entre les niveaux 1 et 3 sont de structurer l'entreprise par la définition de processus, la mise en oeuvre de technologies permettant de supporter ces processus et l'amélioration ainsi que la maîtrise à la fois des processus et des technologies sous-jacentes. En particulier, il s'agit (niveau 3) d'avoir un environnement permettant l'intégration de services avec le minimum d'impacts sur l'architecture globale.

Le niveau 4 dépasse les frontières de l'entreprise pour permettre l'intégration d'applications entre partenaires, entre acteurs de la chaîne logistique. D'où, selon cette perspective,

l'adoption de standards permettant la structuration des échanges et les protocoles de communication.

Chaque changement de niveau se traduit par une plus grande maturité, par une plus grande expertise.

III.3.3 Des stratégies d'intégration

Le modèle client-serveur est particulièrement adapté au processus d'intégration, notamment parce qu'il permet d'implanter plusieurs modes suivant la répartition (ou non) des couches (présentation, logique applicative, gestion des données). En particulier, cela permet d'avoir plusieurs architectures : architectures un-tiers, deux-tiers, trois-tiers, n-tiers. Les organisations peuvent ainsi opter pour le modèle n-tiers permettant de franchir les limites imposées par la logique de l'entreprise (souvent restreinte à ses limites physiques [Abdelmalki 1998]) et permettant de prendre en compte la distribution d'objets, de données de niveaux de granularité différents [Schmidt 2000].

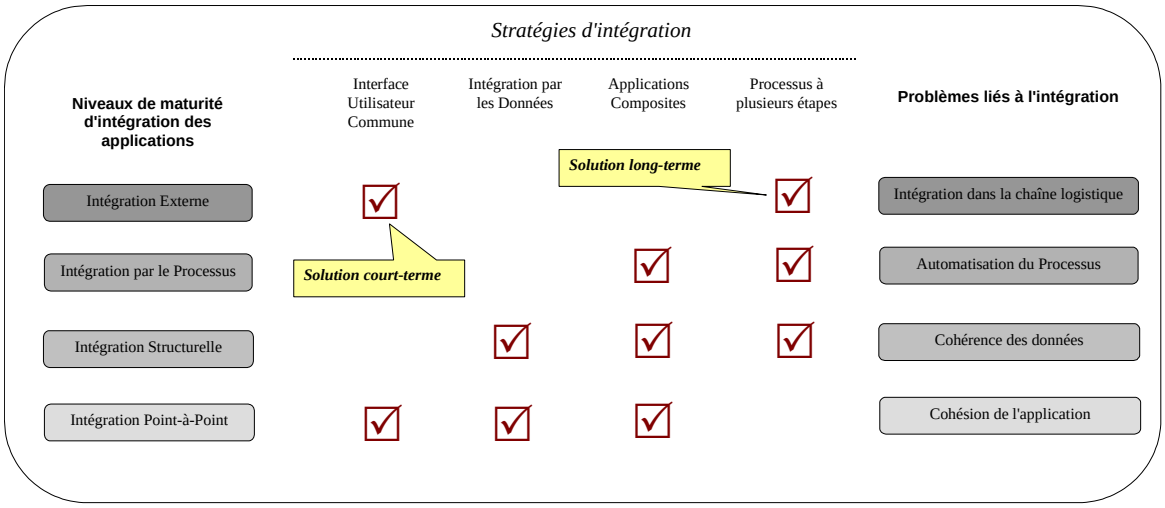

Figure III.13 Plusieurs stratégies d'intégration : la question du choix ...

Quatre stratégies pour l'intégration des applications ont été recensées [Schmidt 2000] (voir Figure III.13) :

a. la stratégie basée sur **une interface utilisateur commune** : cette méthode permet d'intégrer les applications au niveau de la couche "présentation" en fournissant une interface uniforme. Cette approche est largement utilisée pour les applications de l'Internet. Par contre, l'intégration des traitements et des données est souvent laissée à l'initiative de l'utilisateur ;

b. la stratégie basée sur **l'intégration des données** : l'entreprise utilise plusieurs SGBD mais utilise différents outils de manipulation permettant de migrer les données d'un SGBD à un autre. Cette stratégie présente l'avantage de ne pas induire de changement au niveau de la couche présentation ni au niveau de la logique applicative, ce qui permet d'éviter la dépense de moyens financiers (surtout si des applications patrimoines sont identifiées). L'inconvénient majeur est d'introduire des incohérences au niveau des données outrepassant les contraintes d'intégrité et autres règles de cohérence. Certains éditeurs d'ailleurs (comme SAP) refuse leur support technique si l'entreprise utilise cette stratégie ;

c. la stratégie de **composition d'applications** : elle implique une application généralement de taille importante (par exemple, un système PGI[19] – Progiciel de Gestion Intégrée ou le cœur d'un système distribué intégrant l'ensemble des fonctionnalités de base nécessaires à l'entreprise). Par exemple, les modules de SAP, etc. ;

d. la stratégie basée sur **un processus à plusieurs étapes** : cette stratégie utilise une infrastructure intermédiaire (bus logiciel, etc.) comprenant des règles de fonctionnement et permettant d'intégrer la logique application au niveau du modèle de l'architecture de l'application. Cette infrastructure décrit comment les différentes applications doivent interagir (reposant par exemple, sur la notion d'évènements, l'utilisation de moniteurs transactionnels, les courtiers, les moteurs de workflow, etc.

III.3.4 Les modèles d'architectures proposés

L'un des points importants qui caractérise ce domaine est le constat suivant : les entreprises ont recours, autant que possible et pour satisfaire leurs besoins, aux composants issus du marché. Ainsi, un des problèmes auquel elles sont confrontées est l'intégration de ces composants avec les applications patrimoines et autres services de l'entreprises (par exemple, serveur Web, etc.) afin que ces différents systèmes interopèrent.

Les techniciens des entreprises et autres experts en charge de l'intégration des systèmes sont confrontés à deux problèmes :

- le foisonnement de technologies concurrentes facilitant l'intégration (bus logiciels, infrastructures de communication...),
- le comportement de ces technologies qui dévient des comportements attendus.

Les modèles (ou patrons) d'architecture offrent une vision architecturale de l'intégration d'applications en entreprise et constituent de solides fondations aux problèmes d'intégration. Ils permettent d'aider les techniciens et experts à déterminer les meilleurs approches et outils permettant de trouver une solution au problème d'intégration.

Les modèles permettent, en général, d'avoir une certaine flexibilité et surtout une compréhension globale du système. Un modèle d'architecture est défini ainsi [Lutz 2000] :

> il exprime un schéma pour l'organisation et la structuration de systèmes logiciels. Il fournit un ensemble de sous-systèmes prédéfinis en spécifiant leurs relations et incluant les règles et les principes permettant d'organiser les relations entre eux.

L'ensemble des modèles et dérivations reprennent les travaux présentés dans la section III.4.2.

III.3.5 Différentes technologies pour l'intégration

Intégrer différentes applications, potentiellement hétérogènes et qui n'ont pas été conçues à l'origine pour satisfaire les contraintes d'une intégration fait apparaître quelques problèmes [Stonebraker 1999] :

- l'utilisation de chaque application est tributaire de l'environnement (l'entreprise) et fluctue d'une entreprise à une autre ;
- chaque application a sa propre perception des objets (artefacts) de l'entreprise ;

[19] L'expression anglophone équivalente est ERP (Enterprise Resource Planning).

- chaque application peut présenter des recouvrements avec les autres applications avec lesquelles elle doit être intégrée.

Plusieurs options technologiques sont proposées pour intégrer des applications selon ces contraintes (voir Tableau III.1).

Niveau Données	• Moniteurs transactionnels • Serveurs d'application	• Systèmes fédérés de données • Entrepôts de données
Niveau Evènement	• Bus orienté message (Message Oriented Middleware - MOM)	• Publication/Souscription
	Mise à jour (cohérence des applications)	*Lecture (Vues unifiées)*

Tableau III.1 Classification des options pour les EAI [Stonebraker 1999]

L'axe horizontal de la figure concerne les "actions" possibles (mise à jour de donnée ou lecture de donnée) et les problèmes qui se posent : par exemple, l'étiquette "mise à jour" permet de répondre à la question que faut-il faire en cas de mise à jour ? Cette question concerne tout particulièrement la cohérence globale de l'ensemble des applications.

L'axe vertical concerne la manière d'intégrer des applications. Deux applications peuvent être intégrées selon le principe de l'échange de messages. C'est pourquoi il existe un niveau évènement (niveau également appelé intégration par l'application). Un autre moyen permettant l'intégration est le partage d'une base de données commune. Dans ce cas, une application peut écrire de l'information dans la base de données alors qu'une autre va lire les informations dont elle a besoin. Par extension, une application peut être amenée à lire des informations issues de plusieurs bases de données [Stonebraker 1999].

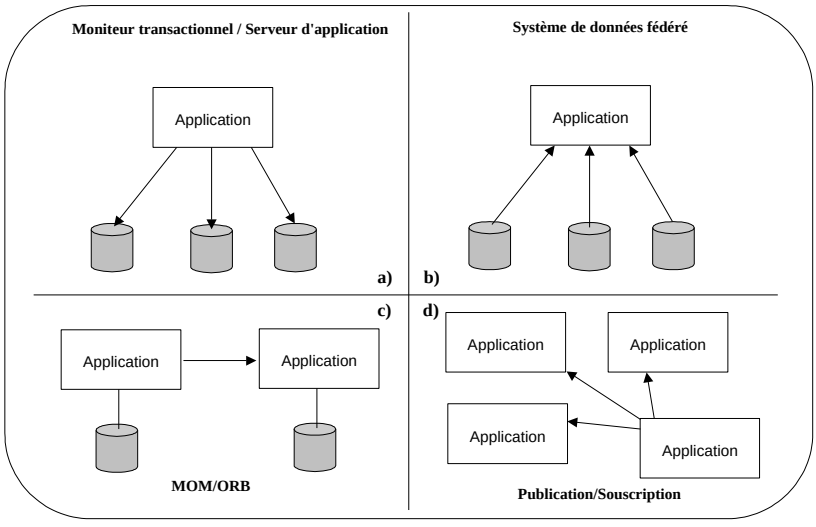

Figure III.14 Modèle opérationnel associé à chacune des approches [Stonebraker 1999]

Chaque alternative est représentée par un modèle opérationnel (voir Figure III.14) expliquant le paradigme de fonctionnement.

Pour résoudre les problèmes de recouvrement par exemple, il faudra utiliser des adaptateurs entre le bus à messages (MOM) ou le courtier d'objets et chacune des applications de façon a convertir un message d'un format d'une application au format d'une autre application. De par

le domaine d'origine de l'auteur (bases de données), la solution qui lui semble la plus adaptée se base sur des systèmes fédérés de données où les données restent accessibles localement et doivent pouvoir être "tirées" (voir Figure III.14.b)) lorsque le besoin s'en fait sentir.

Ainsi, chacune des technologies proposées ci-dessus fait appel aux modèles d'architectures abordés dans la section III.4.2.

La démarche proposée par [Stonebraker 1999] pour intégrer des applications est la suivante :

1. éviter autant que possible de mettre en oeuvre une solution qui n'offre aucun moyen d'opérer des transformations sémantiques. Les transformations simples peuvent être exprimées simplement alors que les transformations complexes sont généralement résolues à l'aide de code (langage de programmation) ;
2. essayer autant que possible d'intégrer les applications au niveau données car les incohérences sont un peu plus faciles à traiter ;
3. identifier la case du tableau (Tableau III.1 et Figure III.14) représentant le cas dominant du problème à résoudre et chercher une solution dans cette case ;
4. ne pas chercher une solution qui aborde l'ensemble des problèmes : elle n'existe pas encore (!) ;
5. éviter autant que possible les produits d'intégration qui ne parviennent pas à garantir l'autonomie locale des applications ;
6. garder à l'esprit que le marché "pousse" pour fournir des systèmes de bases de données fédérés ;
7. utiliser les mécanismes de réplication (de données) à bon escient afin d'atteindre des performances "acceptables".

La règle générale est qu'il n'existe pas de solution universelle et que la solution à un problème posé passe souvent par la mise en collaboration de différents produits permettant l'intégration d'applications.

III.3.6 Evaluation

III.3.6.1 Bilan

Dans les faits, la tendance est clairement vers l'intégration de solutions et d'applications hétérogènes, caractérisée par les EAI [Mann 1999, Hostachy 2000, Schmidt 2000, Beck et al. 2000]. En particulier, les évolutions des différents progiciels de gestion intégrés [MESA 1997a, MESA 1997b], tels que SAP, PeopleSoft, Baan, Oracle, etc. ont montré leurs limites dans le fait d'intégrer l'ensemble des fonctionnalités de l'entreprise au niveau d'une application monolithique. L'heure est à l'intégration d'applications modulaires et modulables reposant sur une infrastructure garantissant certaines propriétés (gestion des transactions, persistance, etc.) [Linthicum 2000].

Les objectifs des EAI sont sensiblement les mêmes que ceux présentés dans cette thèse à savoir, intégrer des applications logicielles hétérogènes et indépendantes, utilisant des technologies parfois incompatibles et qui doivent continuer à être gérées individuellement et de façon indépendante.

III.3.6.2 L'EAI : une infrastructure

La vision considérant l'EAI comme l'épine dorsale de l'entreprise proposée par [Hostachy 2000] permet désormais d'être enrichie avec les considérations techniques et les modèles d'architectures qui viennent d'être abordés.

Dans ce contexte, nous reprenons une vision en couches d'une solution EAI proposée dans [InfoMag 2000] consistant à la définition de trois niveaux :

1. le premier niveau de transport, incluant les passerelles, les connecteurs spécifiques et connecteurs progiciels permettant aux applications d'être intégrées ;
2. le second niveau concerne la transformation et le routage (conséquence d'une mise à jour, etc.) ;
3. le troisième niveau porte sur la modélisation et, entre autre, la gestion des processus métier.

Comme il est mentionné dans [InfoMag 2000], peu de progiciels fournissent des solutions aux problèmes généraux du second niveau. Quant au premier niveau, il souffre de la mise au point de connecteurs (adaptateurs) spécifiques qui sont développés "à la main" pour la plupart. Le troisième niveau n'est abordé par aucun outil d'EAI même s'il semblerait que des travaux passent par l'intégration d'outils de workflow [InfoMag 2000].

III.3.6.3 Positionnement

Nous constatons que les solutions EAI mises en place reposent davantage sur une application de taille importante, telle un PGI sur lequel viennent se connecter d'autres applications. L'approche que nous préconisons vient à l'encontre de cette dernière à savoir, proposer une infrastructure facilitant l'intégration des applications mais indépendante de l'une d'entre elles.

En ce qui concerne les aspects plus techniques, différents articles recensent les alternatives envisageables et disponibles mais aucune démarche/méthode n'a été proposée pour utiliser ces alternatives et les technologies sous-jacentes ; le concepteur de l'intégration a la charge d'identifier les problèmes qui lui sont posés, d'évaluer l'ensemble des modèles d'intégration, d'identifier les variantes selon une approche bas-niveau (il faut mettre en place des technologies). Les solutions et leurs mises en oeuvre reposent aujourd'hui sur l'expérience et le savoir des personnels en charge de l'intégration selon une approche bas niveau.

Nous voudrions, au contraire, exhiber des concepts, mettre à la disposition des concepteurs des moyens les aidant dans leur démarche d'intégration, découpler les problèmes qui se posent (recouvrements, incohérences, etc.) et, enfin, proposer une architecture flexible permettant d'atteindre les objectifs et prendre en compte les contraintes. En ce sens, nos objectifs se situent clairement dans le niveau 4 de la proposition de [Schmidt 2000] (voir Figure III.12), relevant du niveau 3 de [InfoMag 2000], niveaux qui restent vierges de proposition. Nous voulons non seulement proposer une approche permettant de définir "l'épine dorsale de l'entreprise" [Hostachy 2000], mais bien plus encore, une épine dorsale intra et inter-entreprises.

Nous allons, dans la section suivante, présenter des travaux permettant de faire coopérer des composants. Nous trouvons particulièrement les travaux portant sur l'interopérabilité de composant et des travaux portant sur les paradigmes d'interaction de composants issus des Systèmes Multi-Agents (SMA).

III.4 Interopérabilité, Assemblage, Coopération d'outils

L'assemblage des composants en systèmes fait l'objet de nombreux travaux. Le but de l'assemblage, comme le montre la figure III.15 est d'obtenir un système cohérent (une application par exemple) à partir de composants logiciels[20].

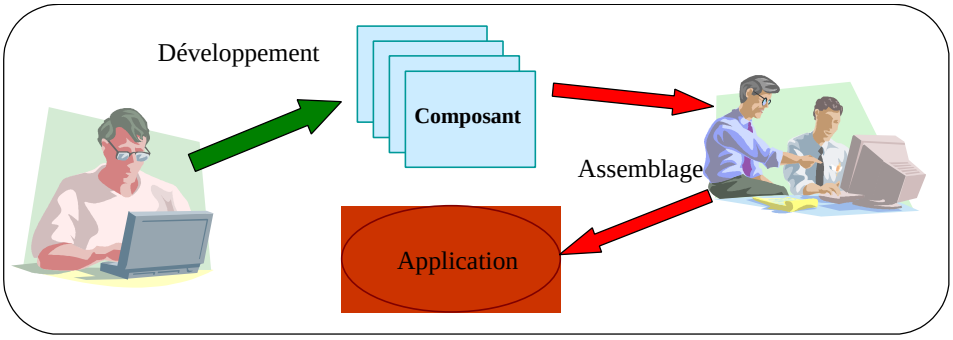

Figure III.15 Programmation par composants

Cette section a pour objectif de balayer certains travaux (ou/et domaines) qui ont porté sur différentes manières d'assembler les composants; en particulier :

1. les travaux les sur les mécanismes d'interopérabilité ;
2. les travaux sur les architectures logicielles et la topologie de systèmes ;
3. les travaux sur la coopération d'agents dans les Systèmes Multi-Agents (SMA).

III.4.1 De l'intégration à l'interopérabilité

C'est dans le rapport Stoneman [Buxton et Druffel 1980] qu'il est mentionné la nécessité de disposer d'une infrastructure pouvant intégrer les outils d'un environnement logiciel. Le rapport faisait état des besoins et de l'architecture d'un environnement afin de fournir un support pour le développement de logiciels écrits avec le langage ADA. Le support proposé se compose d'outils qui interagissent au travers d'un gestionnaire d'objets commun. C'est à la suite de cette proposition que plusieurs travaux ont tenté de normaliser les gestionnaires d'objets. Les plus connus sont les normes PCTE [Thomas 1989, Boudier et al. 1988, PCTE 1993, PCTE 1994, Wakeman et Jowett 1990, Oquendo et al. 1991] et CAIS-A [Oberndorf 1988].

Il est courant de considérer le gestionnaire des objets (comprenant les données et les moyens d'atteindre ces données) comme partie principale des EGLCPs: il s'agit donc bien, à certains égards, un élément intégrateur de l'architecture.

Les travaux portant sur l'intégration des outils ont conduit à la définition de trois modèles conceptuels considérant différents types possibles d'intégration [Wasserman 1989, Wallnau et Feiler 1991] : l'intégration par la présentation, l'intégration par le contrôle et l'intégration par les données.

> L'*intégration par la présentation* permet d'unifier les aspects graphiques des outils. Le point de départ de ce type d'intégration est issu de la constatation qu'homogénéiser les outils permet d'économiser des efforts en terme de développement (modularité et ré-

[20] Chaque approche ou spécifications fait des hypothèses concernant les composants qu'il est possible d'assembler.

> utilisation grâce aux interfaces génériques) et d'utilisation (apprentissage facilité pour l'utilisateur). Ce type d'intégration "inventé" par Apple avec les ordinateurs MacIntosh, est toujours d'actualité (applications MS-Windows, MOTIF, etc.).
>
> L'*intégration par le contrôle* permet aux différents outils de collaborer pour satisfaire un but commun. Les mécanismes qui supportent l'intégration par le contrôle doivent permettre l'exécution à distance de services offerts par les outils, c'est-à-dire, le partage et l'invocation des services entre les outils.
>
> L'*intégration par les données* permet de considérer les données comme pivot entre les outils qui doivent être gérées, à l'échelle de l'environnement (entre les outils), comme un tout cohérent.

En fait, l'intégration d'outils reste une question relativement épineuse, au niveau recherche et bien plus encore, dans le milieu industriel confronté à l'hétérogénéité des systèmes mis en place.

Définition de l'interopérabilité

> **L'interopérabilité** est définie comme étant la capacité d'un ou de plusieurs systèmes ou composants d'échanger de l'information et d'utiliser l'information échangée [IEEE 1990].

Comme nous le remarquons, cette définition couvre des aspects très divers que sont :

- la caractérisation de l'information échangée,
- les moyens permettant les échanges,
- l'exploitation qui est faite de cette information.

L'ensemble de ces aspects concernent les composants et les systèmes logiciels.

Remarque : dans la pratique, la notion d'interopérabilité préserve l'identité des composants interopérant, contrairement au phénomène d'intégration pour lequel les composants perdent leur identité (ils sont "absorbés").

III.4.1.1 Diverses propositions pour interopérer

Cette partie propose un aperçu (non exhaustif mais suffisamment représentatif) de travaux permettant à deux ou plusieurs systèmes d'interopérer.

Le groupe de coalition sur le workflow (Workflow Management Coalition - WfMC) s'est intéressé de près à la question de l'interopérabilité entre différentes applications de workflow comme indiqué dans un document de la WfMC. Nous reprenons ces études pour caractériser les modes d'interopérabilité entre outils [Verjus 1998b, Verjus et al. 2000] (voir Figure III.16) :

- *passage direct de messages entre les outils* : les outils interagissent directement (point à point) ;
- *passage indirect de message* : les produits interagissent par envois de messages vers un "transporteur" ou bus commun (de diffusion) ;
- *passage indirect de messages via une passerelle* : la passerelle (composant intermédiaire) permet de faire communiquer des outils qui utilisent des protocoles de communication différents ;
- *passage de données* : les outils partagent une base de données commune.

III.4.1.2 Les "niveaux" d'interopérabilité

Le groupe de travail de WfMC [WfMC 1995, WfMC 1996a] a identifié huit niveaux d'interopérabilité (voir Figure III.16. - analogie avec les niveaux de maturité – Capability Maturity Model - proposés par [Paulk et al. 1991, Paulk et al. 1993]) en fonction de l'architecture et des caractéristiques opérationnelles des implantations de produits workflow (le produit workflow est ici avec un sens générique : une application destinée au domaine du workflow).

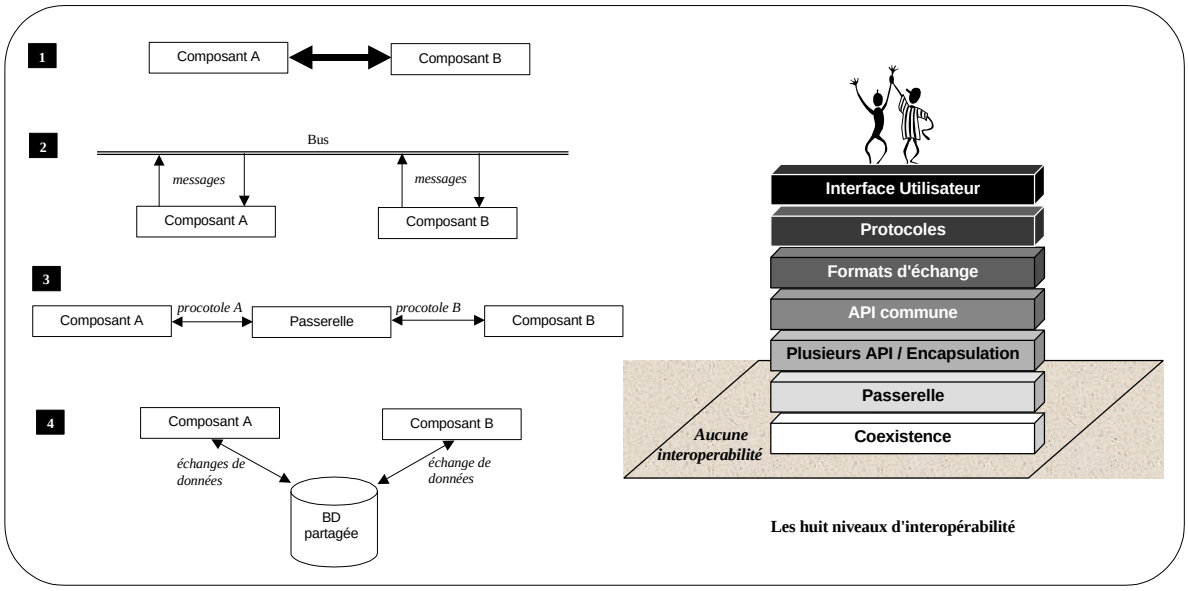

Figure III.16 Les quatre modes et les huit niveaux d'interopérabilité (WfMC – [Verjus et al. 2000, Estublier et Verjus 1999])

Nous notons tout particulièrement le sixième niveau pour lequel les échanges entre moteurs[21] de workflow sont réalisés en utilisant un formant d'échange commun. Ce langage d'échange est défini par la coalition WfMC et s'appelle WPDL (Workflow Process Description Language) [WfMC 1996b] dans lequel le processus est décrit. Les fonctions supplémentaires peuvent être définies en utilisant une extension du langage (comme tout autre langage). Quant au huitième niveau, il stipule que chaque produit workflow "à la norme" WPDL doit se conformer à l'uniformité de ses interfaces avec les utilisateurs (uniformité ergonomique). Ce qui n'est pas réaliste pour des produits issus d'origines différentes.

Un nombre significatif de formalismes d'échange ont vu le jour, et pour des domaines d'application particuliers [Verjus et al. 2000].

III.4.1.3 La gestion du contrôle

La gestion du contrôle prise en compte (ou non) par les mécanismes d'interopérabilité a pour but de permettre aux systèmes (hétérogènes) qui communiquent, d'adapter leur comportement en fonction de l'information échangée [Wasserman 1989]. [Boyer 1994] présente deux types de gestion du contrôle : (1) la notification de l'information (mise à disposition) et (2) l'invocation de services.

1. La notification d'information est généralement réalisée en utilisant la gestion des évènements selon le paradigme publication/souscription ; selon ce dernier, un système est en capacité de publier (émettre) des évènements et d'autres de les recevoir ou non selon qu'ils s'y soient abonnés ou non. Ce paradigme est implémenté par les outils de

[21] Outils, agent, composant, etc. logiciel qui interprète le modèle du processus de workflow.

messagerie (JMS [Sun 1998] par exemple) et divers bus logiciels (services CORBA, etc.).

2. L'invocation de services caractérise l'appel, par un système, d'un service fournit par un autre système. Ce service est souvent une fonction, une procédure ou une méthode. Les interactions offertes par un composant et accessibles sont décrites par ses interfaces (langages OMG-IDL, MIDL, concept d'*interface* du langage Java dans Java/RMI, etc.).

Ces deux types de gestion du contrôle servent de support aux travaux que nous avons déjà évoqués (ingénierie des systèmes à base de composants, EAI en particulier).

III.4.1.4 Evaluation

De nombreux travaux ont porté sur les mécanismes d'interopérabilité et dans des domaines aussi divers que l'industrie aéronautique, les normes pour l'échanges de documents ou de produits industriels, etc. Quelques uns de ces travaux sont présentés dans [Verjus et al. 2000, Estublier et Verjus 1999] et ont fait l'objet d'une étude comparative. Le bilan de ces propositions est le suivant à savoir, qu'aucune des propositions n'offre de réponse satisfaisante à l'ensemble des aspects couverts par la notion d'interopérabilité :

- certaines sont davantage adaptées à la gestion des données (cas fréquemment rencontrés dans le domaine des bases de données),
- d'autres au contraire, offrent des formalismes d'échange dans un domaine d'application particulier mais dont la portée est restreinte,
- alors que le contrôle n'est pas suffisamment abordé.

Si nous regardons les travaux portant sur les échanges d'information, les diverses propositions se sont portées sur l'adoption de formes canoniques pour les types de données et sur des formalismes d'interopérabilité pour les concepts et leurs sémantiques. Il n'en reste pas moins vrai que les concepts ne font ni l'objet de consensus ni ne permettent de couvrir l'ensemble des aspects d'un domaine.

L'arrivée de spécifications émergentes telles que celles du langage XML et des langages dérivés semble conduire à une homogénéisation des formats d'échange, en tout cas pour les données.

III.4.2 Approches et paradigmes permettant l'assemblage d'outils

Cette section présente un état de l'art sur les approches et les paradigmes utilisés pour l'assemblage d'outils logiciels. Ces techniques sont utilisées de façon transversale aux domaines que nous étudions à l'intérieur de ce chapitre (CBSE, EAI, SIF, etc.).

Dans le domaine de l'intégration des applications d'entreprise, cinq modèles d'architecture ont été recensés [Lutz 2000] :

1. **adaptateur** : il convertit l'interface d'une application existante en une interface "souhaitée"[22] ;
2. **messager** : il décrit une approche minimisant les dépendances de communication entre applications ;
3. **façade** : elle fournit une interface simplifiée de l'application, minimisant les dépendances entres les applications clientes et les applications serveurs ;

[22] Cela signifie que l'interface sera transformée de façon à ce qu'elle puisse être utilisée par ailleurs.

4. **médiateur** : il encapsule la logique d'interaction de l'application, minimisant les dépendances ;
5. **automate du processus**[23] : il décrit une approche architecturale minimisant les dépendances entre la logique d'automatisation du processus et les applications.

Il existe, à partir de ces modèles de base, de nombreuses variations [Lutz 2000, Hayden et al. 1999, Woods et Barbacci 1999, Mularz 1995]. Cependant, ces modèles constituent la base des architectures pour l'assemblage d'outils.

III.4.2.1 Le modèle de l'adaptateur

Ce modèle permet, comme son nom l'indique, d'adapter l'interface d'une application dans un format attendu de l'application cliente (celle qui va invoquer les services de l'interface). Ce modèle permet de façon sous-jacente, de fournir une description d'interface pouvant être réutilisée par plusieurs applications clientes (voir Figure III.17).

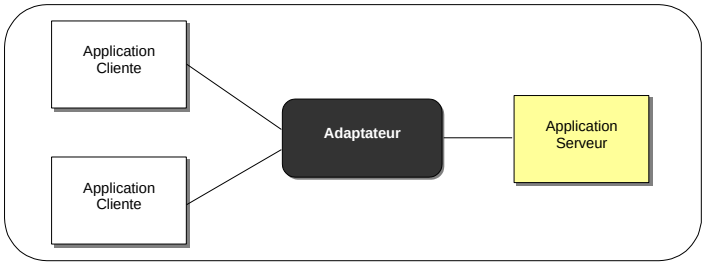

Figure III.17 Principe de l'adaptateur

Les participants à ce modèle d'architecture sont les suivants :

- une ou plusieurs applications clientes ;
- l'adaptateur ;
- l'application serveur.

Selon ce schéma de fonctionnement, les applications clientes invoquent les services de l'application serveur à travers l'interface de l'adaptateur. L'adaptateur convertit l'interface programmable publique (API) exportée par l'adaptateur vers l'API de l'application serveur. L'adaptateur n'a aucune connaissance de l'existence de chaque application cliente. Selon une approche non-intrusive, l'application serveur n'a pas non plus connaissance de l'adaptateur. Selon une approche intrusive, l'application serveur subi quelques modifications.

Evaluation : l'atout majeur de ce modèle d'interaction est la réduction de la charge concernant les messages entre plusieurs clients et plusieurs fournisseurs car il est alors plus intéressant d'avoir un agent "pivot" centralisant les requêtes (plutôt que chaque agent client ait à gérer une liste de fournisseurs potentiels) et permettant les conversions (entre requêtes et demandes de services et vice-versa).

Le modèle de l'adaptateur présente deux déclinaisons importantes dans le domaine de l'intégration d'applications d'entreprise [Lutz 2000] : (1) la façade de l'application et (2) l'adaptateur de courtier de message.

La façade exporte les services d'une application à d'autres applications à travers une interface publique. Cette approche est largement utilisée pour exporter les services d'une application patrimoine.

[23] On notera l'appellation différente du "moteur de processus" (voir deuxième chapitre)

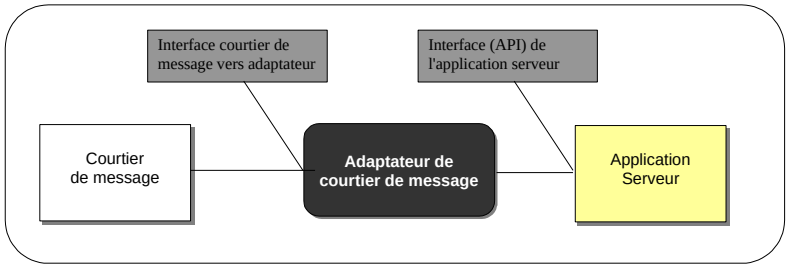

Figure III.18 Principe de l'adaptateur de courtier de message

L'adaptateur de courtier de message (Figure III.18), très proche de la façade, diffère cependant de par le fait que l'interface publique de l'adaptateur est telle que l'a définie un courtier de message. Un courtier de message est un composant intelligent intermédiaire qui dirige le flux des messages entre les applications, considérées comme sources et consommateurs d'information. Un courtier de message a habituellement plusieurs adaptateurs en fonction des types de systèmes avec lesquels il interagit.

Nous trouvons dans la littérature différentes déclinaisons de ce modèle que l'on nomme "ambassadeur", "facilitateur", etc.

III.4.2.2 Le modèle du messager

Ce modèle est particulièrement utilisé lorsqu'il n'est pas nécessaire ou lorsqu'il est impossible de découpler la logique d'interaction de l'application des applications elles-mêmes. Le bénéfice est que les dépendances en terme de communication entre les applications sont minimes. cela permet d'avoir une approche plus flexible. Ce modèle permet d'implanter les modèles de communication suivants :

- un-à-un en mode synchrone (question/réponse) : un client unique effectue une requête auprès d'un serveur et attend la réponse de ce dernier ;
- un-à-un en mode asynchrone (gestion d'une queue de messages) : un client unique effectue une requête auprès d'un serveur. Contrairement au modèle précédent, le client continue son exécution (il ne suspend pas son exécution en attendant la réponse du serveur) ;
- un-à-plusieurs en mode asynchrone (publications et souscriptions). Un client et un ou plusieurs applications serveurs.

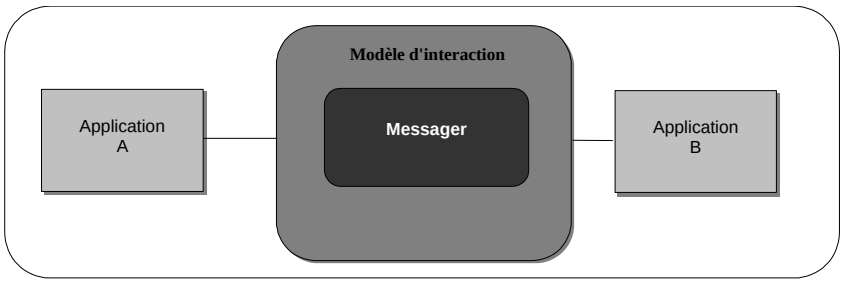

Figure III.19 Principe du messager

Les participants à ce modèle sont les suivants (voir Figure III.19) :

- les applications qui doivent être intégrées (A et B),
- le composant messager, responsable du routage des messages entre les applications et garantissant la transparence de la localisation des services.

Le modèle d'interaction décrit le modèle utilisé par les applications pour interagir entre elles. Trois types de modèles d'interaction ont été recensés [Lutz 2000] :

- l'invocation de méthode à distance ;
- la gestion d'une queue de messages ;
- publication/souscription.

Selon ce modèle, les applications sont elle-mêmes responsables d'implémenter la logique d'interaction. Le messager est totalement étranger à la sémantique d'interaction des applications.

Une approche relativement flexible consiste à combiner le modèle du messager avec le modèle de l'adaptateur.

III.4.2.3 Le modèle de la façade

Ce modèle a pour principal objectif de permettre l'intégration d'applications clientes avec des applications serveurs. Ce modèle permet la réutilisation et la flexibilité des applications et des services permettant l'intégration.

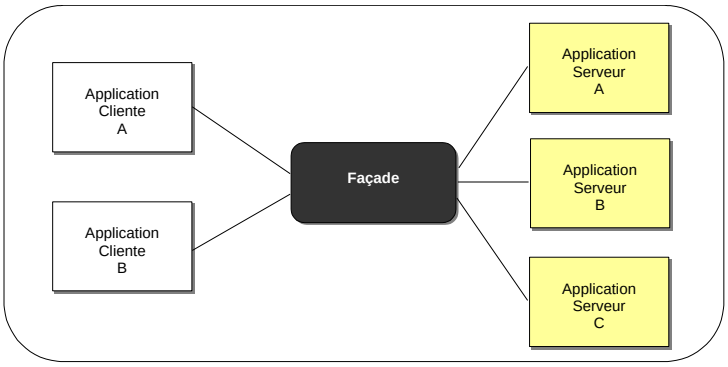

Figure III.20 Principe de la façade

L'idée de la façade est de fournir une interface simplifiée à une ou plusieurs applications clientes (voir Figure III.20). Les participants à ce modèle sont :

- une ou plusieurs applications clientes,
- le composant de façade,
- une ou plusieurs applications serveurs.

Les applications clientes invoquent les services "simplifiés" de la façade. Cette dernière est une abstraction des fonctionnalités des serveurs facilitant l'utilisation des fonctionnalités des serveurs. Elle transforme son interface vers les interfaces des serveurs. Le modèle d'interaction utilisé est initié, soit par le client, soit par un client, soit par un serveur ; cela veut dire en fait, que l'interaction est unidirectionnelle : la façade ne connaît pas l'existence de chacune des applications clientes et les applications serveurs n'ont que faire du composant de façade.

La façade est donc un composant qui opère un découplage entre les clients et les serveurs en fournissant une abstraction de ces derniers. Cela permet de découpler les services de leurs implémentations offrant une interface de plus haut niveau.

Evaluation : permet d'incorporer des applications patrimoines, existantes. L'agent façade permet les transformations entre l'ontologie de l'application et l'ontologie du système et vice-versa. La limitation principale est la lourdeur introduite par les procédures de conversions.

III.4.2.4 Le modèle du médiateur

L'originalité de ce modèle d'intégration réside dans le fait qu'il propose une architecture où la logique d'interaction est encapsulée et découplée des applications participantes. Les avantages majeurs de cette approches sont les suivants :

- la maintenance est simplifiée du fait que la logique d'interaction est centralisée au niveau du composant médiateur (et non distribuée sur chaque application) ;
- les services réutilisables peuvent être correctement construits autour de la logique d'interaction ;
- les dépendances et les différences avec d'autres applications (patrimoines, etc.) sont minimisées.

Ainsi, cette approche est relativement flexible et contrairement à l'approche précédente (celle de la façade), le composant médiateur a la connaissance de l'application.

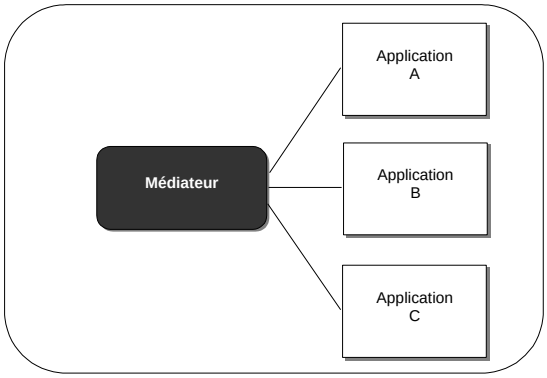

Figure III.21 Principe du médiateur

Les applications interagissent directement avec le composant médiateur au lieu d'interagir avec les autres applications. Cette approche offre plusieurs degrés dans la gestion et le contrôle de l'interaction. Ils peuvent être schématiquement regroupés en deux catégories, (1) les approches par état, (2) les approches sans état.

Un médiateur "sans état" est utilisé lorsque la logique intrinsèque du médiateur ne dépend pas de l'état des interactions entre applications, c'est-à-dire, que la réponse du médiateur à un message (ou à un évènement) émanant d'une application dépend seulement du contenu du message et est donc indépendante des interactions précédentes et de leur ordonnancement. Cette approche permet de faire les transformations nécessaires (syntaxiques et/ou sémantiques) sur le contenu des messages (d'un format de message vers un autre) ainsi que la logique du routage des messages.

Un médiateur "à état" est utilisé lorsque la logique d'interaction intrinsèque au médiateur dépend des interactions précédentes. Par exemple, cette approche est illustrée lorsque le médiateur accumule les évènements et déclenche l'envoi d'un message après vérification d'une condition portant sur les différents évènements accumulés. Cette approche exige bien souvent une gestion des états et un système de persistance des états.

La gestion de la persistance permet de recouvrir un état cohérent du système après des dysfonctionnements du médiateur. Cependant, cela s'accompagne d'algorithmes parfois complexes.

Parmi les paradigmes d'interaction basés sur le modèle du médiateur, le courtage, largement utilisé, fonctionne selon trois étapes [ODP 1995, Wolisz et Tschammer 1993] :

- l'exportation des offres de services : les fournisseurs exportent leurs offres de services vers les courtiers ;
- l'importation de services: les clients envoient leurs appels d'offres aux courtiers ;
- la recherche et la sélection : le courtier recherche parmi les offres qu'ils a reçues la ou les plus appropriée(s) aux besoins exprimés par le client. Le courtier peut soit retourner la liste des offres qu'il a préalablement sélectionnées au client (ceux-ci peuvent alors directement interagir avec les fournisseurs), soit envoyer l'appel d'offre du client aux fournisseurs qui sont en capacité de le satisfaire.

Evaluation : le modèle du "médiateur" propose, en plus de courtage, un modèle de comportement et d'interaction (appelé modèle de coordination) inclus dans les modèles de conversations et basé sur les accointances entre agents, dans les SMA. Contrairement au modèle de façade qui ne fournit qu'un protocole unidirectionnel, le médiateur permet des interactions multidirectionnelles. L'un des avantages de ce modèle est le découplage entre les agents/applications "collègues", réduisant les interconnexions directes. L'une des limitations est le risque de saturation si un nombre important d'agents/d'applications communiquent avec un même médiateur en même temps. Il est possible, dans un système, d'avoir plusieurs "couches" (auxquels cas il y aurait des médiateurs de médiateurs...).

III.4.2.5 Le modèle de l'automate du processus

Ce modèle a principalement pour but de minimiser les dépendances entre la logique de contrôle de l'automate du processus et les systèmes des technologies de l'information ; en effet, le concept d'activité fait que l'ensemble des interactions entre systèmes (et les interactions entre personnes) sont cachées du contrôleur de processus.

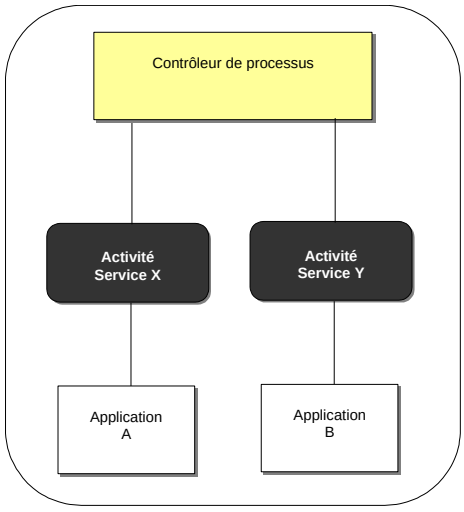

Figure III.22 Principe du moteur de processus

Les avantages d'une telle approche sont :

- des moteurs de processus différents peuvent être implémentés de façon relativement économique ;
- l'architecture fournit une analyse des processus qui n'a pas son équivalent ("goulots d'étranglements", statistiques, utilisation des ressources, etc.) ;
- l'architecture offre une flexibilité inégalée dans la redéfinition et le déploiement d'applications centrées processus ;

- l'implémentation des applications centrées processus est alignée sur la vision du gestionnaire de projet (manager) ; ainsi, l'écart entre la vision du manager et l'implémentation informatique de cette vision est considérablement réduit ce qui permet de réduire les erreurs potentielles de transformations entre les besoins et les solutions informatiques censées satisfaire ces besoins ;
- comme les modèles de façade et du médiateur, la logique d'intégration est encapsulée et peut être partagée.

Ce modèle fait l'objet des travaux présentés dans le second chapitre de cette thèse.

III.4.2.6 Evaluation des modèles

En reprenant l'ensemble des modèles d'architecture, nous pouvons en retirer les points suivants :

- le modèle de l'adaptateur permet de convertir l'interface d'une application vers une interface attendue d'une application cliente ;
- le modèle du messager permet de minimiser les dépendances de communications directement entre les applications ;
- le modèle de la façade offre une abstraction des services entre le client et le serveur, découplant le serveur de son implémentation ;
- le modèle du médiateur permet d'encapsuler la logique d'interaction entre applications ;
- le modèle de l'automate de processus permet de découpler la logique du processus et les technologies.

III.4.2.7 Les architectures

Nous retenons, des travaux de recherche précédemment présentés, deux catégories d'architectures caractérisant le degré de liberté des agents, des applications :

- une architecture hiérarchique dans laquelle les agents sont soumis à une autorité commune ;
- une architecture point-à-point dans laquelle les agents sont très autonomes et gèrent, à leur niveau, la coordination du système.

Dans [Hayden et al. 1999], les auteurs placent les architectures fédérées dans la catégorie des architectures hiérarchiques dans le sens où le concept de fédération introduit la notion de fonctionnement commun (cette notion peut couvrir plus ou moins de choses suivant les systèmes et les domaines d'application).

III.4.3 La coopération d'agents dans les SMA

Plusieurs travaux de recherche se sont intéressés à l'interaction de composants. On trouve dans la communauté des systèmes multi-agents (SMA) différentes propositions portant sur le contrôle de la coopération d'agents en vue de résolution de problèmes.

C'est dans [Alloui 1996] que nous trouvons les définitions et la justification de plusieurs termes qui sont également largement utilisés (de façon implicite ou explicite) dans le cadre de fédérations de composant. Entre autre [Alloui 1996] :

- le terme "coordination est utilisé pour signifier l'agencement d'un tout selon un plan logique pour une fin déterminée, arrangement, organisation, synchronisation, disposition, enchaînement" ;
- le terme "coopération" est utilisé pour signifier collaboration, aide, accord, appui, concours, contribution".

Selon notre problématique, les composants que nous considérons dans le cadre des fédérations doivent :

- éventuellement être coordonnés pour atteindre les objectifs de la fédération selon un plan déterminé,
- coopérer pour atteindre ces objectifs (car un composant seul ne peut le faire).

III.4.3.1 La coordination dans les approches issues de l'Intelligence Artificielle Distribuée

Différents travaux se sont intéressés à la coopération et à la coordination d'agents. En particulier, les travaux proposés dans Peace+ [Alloui 1996, Alloui et Oquendo 1996a, Alloui et Oquendo 1996b] où les agents sont des agents de processus, les travaux de [Sichman 1995], qui mettent en place les relations de dépendances entre les agents afin d'accomplir les tâches qui leur sont demandées. [Allouche 1998] reprend les travaux de Sichman et ajoute le concept d'organisation des agents. D'autres encore se sont intéressés à l' "auto-organisation" de groupes d'agents, c'est-à-dire, la capacité d'un système à s'auto-structurer par l'allocation de tâches. Les comportements des agents régis par des règles permettent de garantir à la fois l'autonomie des agents mais aussi la cohérence globale du système [Malville et Bourdon 1998, Malville 1999].

En fait, nous remarquons qu'il existe différentes terminologies (coordination et organisation) qui ne traduisent pas forcément de véritables nuances sémantiques et qui ont fait l'objet de langages permettant la coordination, la coopération, etc. Nous pouvons ajouter que l'utilisation de tel ou tel terme est bien souvent conditionné par :

- l'existence (ou non) d'une *entité intermédiaire* entre les composants (appelée médiateur, courtier, "facilitateur" [FIPA 1998], coordinateur [Andreoli et al. 1998, Andreoli et al. 1999], etc.). Cette entité intermédiaire peut être en charge (1) des échanges d'informations entre composants (flux de données), ou/et (2) de contrôler les actions des composants (flux de contrôle) ;
- le degré d' "*intelligence*" des agents, à savoir, (1) leur capacité à réagir ou agir selon des modèles qui leurs sont propres [Alloui 1996] ou/et (2) leurs connaissances [Sichman et al. 1994, Sichman et Demazeau 1994] sur les objectifs qu'il faut atteindre et les moyens d'y parvenir (connaissances sur eux-mêmes et sur les autres agents de l'environnement) en vue de constituer des *plans d'action* [Sichman et Demazeau 1994]. De nombreux travaux considèrent la capacité des agents pour en déduire des "raisonnements" et ainsi, être en mesure de s'adapter (affinités entre agents, évaluation des dépendances et reconfiguration). Nous notons, dans cette mouvance, les travaux portant sur les fédérations de courtiers [Lee et Benford 1995]. Nous retrouvons également la notion de groupes d'agents constituants des coalitions en vue de résoudre des tâches [Schiman 1995, Malville 1999].

D'autres approches s'appuient sur la notion de contrats entre agents, de négociations, de lois de marché, etc. [Smith 1980, Stonebraker et al. 1994b, Malone 1987, Malone et al. 1988]. Ces propositions proviennent, pour la plupart, du domaine de l'Intelligence Artificielle Distribuée (IAD) et sont mises en applications dans les systèmes multi-agents.

On trouve également des travaux similaires dans les approches du groupe FIPA[24] [FIPA 1998] dont l'un des objectifs est de fournir une plate-forme pour la coordination d'agents intelligents. FIPA fournit des spécifications qui maximisent l'interopérabilité d'agents pouvant provenir de différents développeurs en définissant des interfaces pour différents composants d'un environnement logiciel. Ces interfaces interposent des humains, des agents, des logiciels non agents et le monde physique.

L'approche consiste à faire coopérer des agents pour atteindre des objectifs individuels et ou collectifs avec prise en compte de systèmes déjà existants (*legacy agent systems*). Pour cela, FIPA propose un langage formel *"Agent Communication Language"* (ACL) fondé sur les "actes communicatifs" (*communicative acts*) [FIPA 1997]. La sémantique formelle apportée est celle de la théorie des actes de langage (*speech acts theory*) qui part du principe qu'une communication est une action qui s'accomplit dans un contexte, qui a des effets et un but (par exemple : *inform, confirm, refuse, etc.*). FIPA spécifie un ensemble d'actes de communication en termes de types de messages, de leur objectif et des effets escomptés sur les agents impliqués dans la communication. La sémantique formelle est fondée sur une logique modale (*belief*, *intention*, *uncertain*). ACL se situe à un plus haut niveau d'abstraction en comparaison avec CORBA par exemple. En utilisant les actes de communication spécifiés, FIPA propose aussi des protocoles d'interaction de haut niveau (requête, réseau de contrat, etc.). D'autres protocoles peuvent être explicitement définis.

Les données manipulées sont transportées dans les actes de communication. Elles sont exprimées en termes d'objets appartenant à des *ontologies*. Une ontologie définit le domaine du discours et la sémantique à laquelle la donnée appartient. Les ontologies sont définies séparément des actes de communication qui y font référence. Elles peuvent être écrites dans n'importe quel langage.

Le contrôle entre agents est assuré à travers les protocoles d'interaction. Ceux-ci définissent l'enchaînement des actes de communications, lesquels sont interprétés de manière indépendante par les agents. Un protocole est spécifié en termes de réseau d'actes de communication où à chaque acte de l'agent expéditeur sont reliés les actes possibles de l'agent destinataire. FIPA dispose d'un langage de communication de haut niveau indépendant des produits qu'il manipule. La communication entre agents est définie formellement, ce qui empêche toute ambiguïté.

Les agents (clients) peuvent interagir avec des systèmes logiciels "externes" (applications patrimoines, etc.) à travers la définition d'une facette (voir Figure III.23).

[24] Foundation for Intelligent Physical Agent, http://www.fipa.org

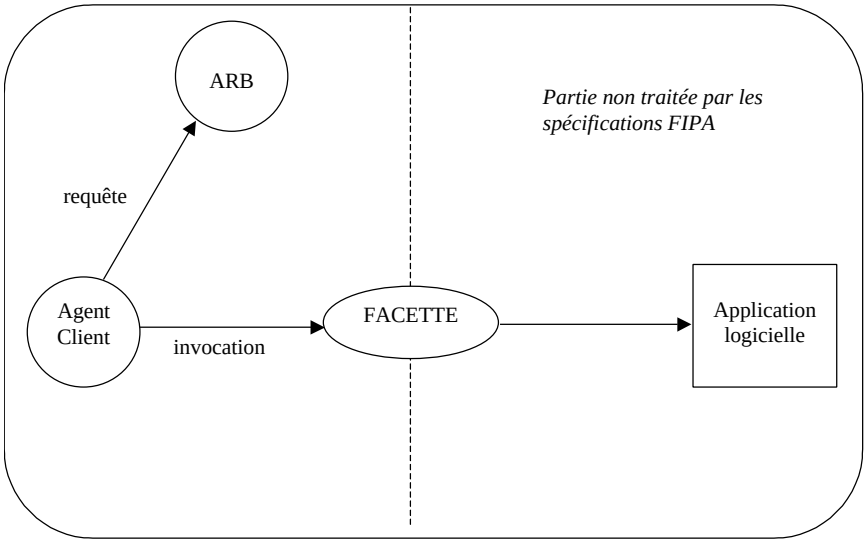

Figure III.23 Schéma général d'une intégration d'un agent logiciel [FIPA 1997]

Le courtier d'agents (ARB pour Agent Ressource Broker) fournit aux agents intéressés, un ensemble de descriptions logicielles à partir duquel il est possible de connaître quels sont les services disponibles. La façon dont la facette interagit avec l'application logicielle n'est pas spécifiée et, est à la charge du développeur.

III.4.3.2 Systèmes à base de lois : du contrôle pour les agents

Ce sont les travaux de Minsky [Minsky 1991] sur les Systèmes Gouvernés par les Lois[25] (SGL), qui présentent la définition d'un modèle de protocole à base de lois. Selon ce modèle, un protocole est défini par un ensemble de "lois" ou contraintes décrivant comment les composants doivent interagir en fonction de l'état du système (cet état est directement lié à l'enchaînement d'actions).

Le modèle abstrait d'un SGL est :

- un système qui est vu comme un ensemble d'objets (composants) interagissant entre eux par le biais de messages ;
- une loi est définie par un ensemble de règles concernant l'échange de messages entre les objets du système (les règles sont également des objets du systèmes) ;
- la présence de composant faisant appliquer les lois.

Un objet est un couple (état, agent) ; un état est un ensemble de valeurs qui ne peuvent changer uniquement par réception de messages.

Chaque composant est contrôlé par un composant contrôleur en charge de faire appliquer les lois (aucune action ne sera permise si une loi l'interdit)[26]. A chaque agent (composant) du système est associé un composant contrôleur qui s'intercale entre l'agent et le reste du système. La description d'une loi qui implique plusieurs agents est dupliquée au niveau de chacun de leur contrôleur respectif, de telle sorte que le contrôle assurant la coordination est totalement décentralisé [Minsky et Ungureanu 1997].

Les notions de droits des agents est présente implicitement : chaque agent est soumis (ou non) à un ensemble de lois qui va déterminer sont comportement. Cependant, dans un tel environnement, les agents sont supposés avoir la capacité d'intégrer le système à base de lois

[25] Le terme anglophone utilisé dans la littérature est "Law Governed Systems".
[26] Le composant qui fait appliquer les lois est appelé dans la littérature, "enforcer" [Minsky 1991]

et, entre autre, pouvoir interagir avec le contrôleur qui lui est associé. L'implémentation du système propose un mécanisme de contrôle dans lequel rien ne peut se faire à l'insu de l'agent (puisque l'ensemble des messages émis et reçus transitent de facto et explicitement par le contrôleur). Cette approche basée sur la définition de lois implémente le modèle à base de "contrôleur" que l'on a présenté précédemment.

III.4.3.3 Evaluation

Les SMA distribués ont permis d'étudier et d'apporter différents paradigmes d'interaction, de coordination qui sont utilisés ou qui servent de base aux modèles d'architectures des EAI. Cependant, les approches dans le domaine des SMA ont pour principale caractéristique d'être intrusives, c'est-à-dire, qu'elles ne font aucune (ou peu) hypothèse sur les agents : pour la plupart des systèmes, les agents sont développés (ou doivent être modifiés) pour s'adapter à l'environnement dans lesquels ils évoluent, particulièrement aux protocoles mis en place. Les capacités à priori des agents pour de tels systèmes ne peuvent être banalisées à l'ensemble des systèmes à base de composants...Particulièrement à base de composants existants. De nombreux modèles d'architectures reprennent, voire complètent, les travaux présentés dans la section III.4.2.

Il résulte de ces remarques que l'organisation des systèmes est basé sur la capacité (1) des agents eux-mêmes et (2) d'agents intermédiaires à "raisonner" pour s'adapter aux demandes de services des uns et des autres. Il s'agit là d'hypothèses relativement contraignantes. D'autre part, les paradigmes mis en place (lois du marché, contrats, etc.) ne sont pas interchangeables d'un système à un autre. Les spécifications FIPA fournissent un certains nombre de possibilités pour définir les interactions entre agents. Cependant, ces agents ne sont pas des composants logiciels autonomes "quelconques" [FIPA 1997].

Ces approches présentent quelques inconvénients majeurs à nos yeux qui nous font dire que ces approches ne sont pas adaptées à la problématique que nous étudions :

1. hypothèses fortes sur le comportement interne du composant,
2. le concept d'autonomie des agents inclut leur capacité de raisonnement [Allouche 1998], la connaissance réciproque,
3. rigidité du mode de fonctionnement (pour la plupart),
4. approches généralement intrusives.

La section du document qui va suivre va traiter des approches de fédération de bases de données et de systèmes de workflow.

III.5 Vers des fédérations

III.5.1 Les systèmes fédérés

Le terme "fédération" est défini comme étant une union, une alliance d'entités. Par extension, il s'agit d'une "association de plusieurs sociétés, [...], groupes, sous une autorité commune" [Robert 1989].

> Nous définissons donc un système fédéré comme étant une association d'entités logicielles placées sous une autorité commune.

Cette définition suppose donc de caractériser :

1. les entités[27] logicielles ;
2. la gestion de la communauté.

Au cours des dernières années, différentes communautés ont travaillé plus ou moins explicitement sur la notion de fédérations : les bases de données fédérées, les systèmes d'information fédérés, les outils de workflow fédérés, les environnements de génie logiciel centrés processus, etc.

Nous allons, dans un premier temps, étudier quelques exemples de systèmes fédérés puis, allons axer notre étude sur les EGLCP.

III.5.1.1 Introduction

Les différentes entités de la fédération sont, pour nous, des entités logicielles appelées souvent composants ou outils. L'ensemble de ces composants entrent en ligne de compte dans le but de la fédération et sont amenés, ensemble, à atteindre ce but. Ainsi, les traitements de la fédération sont, en fait, répartis au niveau des différents composants de cette dernière. L'un des enjeux est donc de gérer à la fois l'autonomie des composants avec, bien souvent, leur nécessaire synchronisation [Estublier et Verjus 1999].

Plusieurs points sont ainsi à étudier :

- l'autonomie ;
- l'hétérogénéité ;
- la distribution.

III.5.1.2 L'autonomie

Dans le contexte des systèmes fédérés, une classification a été proposée [Özsu et Valduriez 1999] :

- **autonomie de conception** : signifie qu'un composant est indépendant d'autres composants dans sa conception intrinsèque (considérant son univers de discours, son modèle de donnée, son modèle d'exécution, etc.). L'autonomie de conception considère également le fait de pouvoir unilatéralement modifier la conception du composant à tout instant; cet aspect à des incidences considérables sur les infrastructures reposant sur la coopération de composants ;
- **autonomie de communication**[28] : signifie qu'un composant peut unilatéralement décider avec quels composants de l'infrastructure il peut communiquer. Considérant la problématique des fédérations, cela se traduit par le fait que les composants peuvent entrer et sortir de la fédération à tout moment ;
- **autonomie d'exécution** : signifie l'indépendance du composant pour exécuter et ordonnancer les invocations qu'ils reçoit de l'extérieur. Notons toutefois que cette

[27] Comme nous avons pu le constater dans le document, le concept d'entité ici recouvre les notions de composant, de partie, d'agent, de système, d'outils, etc.

[28] Nous identifions deux dimensions à l'autonomie de communication : (1) un point de vue interne considérant avec quels composants de la fédération le dit composant décide de communiquer et (2) un point de vue externe, à savoir avec quels composants (indépendamment de ceux de la fédération) y compris la fédération elle-même le composant décide de communiquer. Seul le point de vue externe est ici étudié.

autonomie ne peut être préservée lorsqu'une gestion transactionnelle du système est mise en place.

Nous trouvons parfois dans la littérature l'autonomie d'association; nous pensons qu'elle n'est qu'une combinaison de l'autonomie de conception et de l'autonomie de communication.

III.5.1.3 L'hétérogénéité

L'hétérogénéité est inhérente au fait que des composants autonomes peuvent être développés selon des approches, des méthodologies différentes, etc. (de la même manière qu'un artefact du monde "réel" peut être modélisé différemment). L'intégration de composants requiert donc de gérer l'hétérogénéité entre les composants (par la construction de "ponts").

Plusieurs catégories qualifiant le concept d'hétérogénéité ont été recensées [Wiederhold 1993] : (1) l'hétérogénéité syntaxique, (2) l'hétérogénéité du modèle de données et (3) l'hétérogénéité logique.

Hétérogénéité syntaxique

- *L'hétérogénéité technique* couvre différents aspects techniques comme les plates-formes d'exécution et les systèmes d'exploitation, les méthodes d'accès (protocoles, méthodes de connexion, procédures de sécurité, etc.)
- *L'hétérogénéité des interfaces* existe si différents composants sont sollicités au travers de langages différents. Il n'est pas question de choix technique (comme un pilote JDBC, etc.) mais bien du choix des méthodes d'accès :
    - hétérogénéité des langages: différents langages de requêtes avec des restrictions potentielles ;
    - restrictions de requêtes (requêtes autorisées ou non, expression d'un nombre restreint de conditions, jointures limitées, etc.).

Hétérogénéité du modèle de données

L'hétérogénéité des modèles de données exprime le fait que différents modèles de données associent des contenus sémantiques différents à leurs concepts. Par exemple, un modèle relationnel n'a pas de concept d'héritage contrairement à un modèle orienté-objet. Il y a alors des concepts qui sont différents et qui ne sont pas couverts par l'ensemble des modèles de données des différents composants, comme des concepts qui n'ont pas la même signification selon le composant et le modèle de données qui lui est associé. Bien que l'hétérogénéité du modèle de données est un cas particulier d'un cas d'hétérogénéité sémantique, il est souvent à part dans un nombre important de système (dû à la place prépondérante des données d'un système).

L'hétérogénéité logique

- *L'hétérogénéité sémantique* concerne la sémantique associée aux données et aux modèles. Cette hétérogénéité exprime le fait que le nom symbolique d'un concept peut être interprété de façon différente suivant les systèmes. Par analogie, l'interprétation de noms par différentes personnes peuvent ne pas coïncider (représentations mentales différentes). Ces conflits sémantiques se produisent lorsque (1) un même nom symbolique recouvre différents concepts (on parle d'homonymie), ou alors (2) que plusieurs noms symboliques recouvrent le même concept (synonymie).

Le contenu sémantique des valeurs d'une donnée est défini au travers d'un élément du schéma selon lequel ces valeurs sont présentées. Les valeurs ne comportent pas explicitement un contenu sémantique mais sont définies en totalité par la portion de schéma à laquelle elles se rapportent.

- *L'hétérogénéité schématique* concerne l'encodage des concepts par différents éléments du modèle de données. Par exemple, dans le modèle relationnel, trois types de conflits ont été recensés [Krishnamurthy et al. 1991, Miller 1998] :
    - relation ↔ nom de l'attribut,
    - nom de l'attribut ↔ valeur de l'attribut,
    - relation ↔ valeur de l'attribut.

  Un exemple pour illustrer le conflit "nom de l'attribut ↔ valeur de l'attribut" est représenté par la figure suivante:

Nom Prof.	**Logique**	**BD**	**IA**
Jean T.	X		
Denis F.	X	X	
Estelle G.			X

Nom Prof.	**Cours**
Jean T.	Logique
Denis F.	Logique
Denis F.	BD
Estelle G.	IA

Figure III.24 Conflit "nom de l'attribut « valeur de l'attribut"

Alors que la table de gauche représente les cours enseignés par un professeur comme des noms d'attributs, la seconde table (de droite) représente les cours comme des valeurs de l'attribut "Cours".

- *L'hétérogénéité structurelle* est recensée lorsque des éléments identiques ayant le même sens (contenu sémantique identique), modélisés avec le même modèle de données sont schématiquement homogènes mais structurés de différente façon (par exemple des attributs regroupés dans des tables différentes).

III.5.1.4 La distribution

La question de la distribution physique des sources de données est orthogonale à l'autonomie et à l'hétérogénéité des systèmes. Depuis déjà quelques années, il est naturel de penser que les données et les traitements ne soient pas physiquement sur un même lieu ou machine mais, au contraire, répartis sur un réseau.

Les techniques (CORBA OMG, Java/RMI, COM/DCOM, etc.) permettant cette distribution ont été présentées dans ce document (voir section III.2.1). Cependant, nous allons voir dans la suite qu'une classification des systèmes peut se faire selon les dimensions que nous venons de présenter.

III.5.1.5 Classification des systèmes

Cette classification est établie selon la distribution et l'hétérogénéité [Busse et al. 1999] :

- un *système unique* (monolithique, centralisé) s'exécute comme une application monolithique, sur une seule machine. Les données (persistantes) gérées par un tel

système sont, soit stockées dans un SGBD, soit dans des fichiers du système de fichiers de la machine hôte ;

- un *système distribué* (SD) pour lequel les données sont physiquement distribuées sur plusieurs sites interconnectés ;
- un *système hétérogène* (SH) qui est une collection de systèmes qui diffèrent selon des aspects logiques ou syntaxiques (plates-formes, système d'exploitation, modèle de données, etc.).

Si nous ajoutons la dimension de l'autonomie (voir section III.5.1.2), nous abordons la définition d'un *système fédéré* (SF), vu comme un ensemble de composants (ou systèmes) distincts et indépendants : les participants de la fédération.

III.5.1.6 Les systèmes d'information fédérés

En général, les systèmes d'information fédérés (SIF) respectent une architecture trois-tiers (voir Figure III.25). Les applications et les utilisateurs accèdent à un ensemble de sources hétérogènes de données à travers un niveau (ou couche) intermédiaire qui est un composant logiciels qui proposent une manière uniforme d'accéder aux données stockées dans les différentes bases.

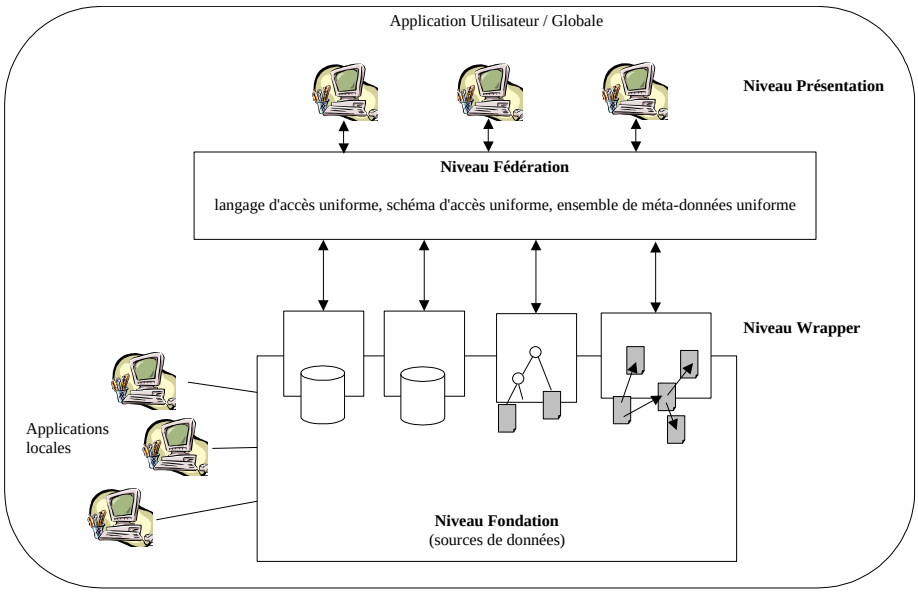

Figure III.25 Architecture pour un système d'information fédéré (d'après [Busse 1999])

L'uniformité est atteinte grâce à une stratégie d'interopérabilité qui propose un schéma fédérateur, un langage de requêtes uniforme, un ensemble de méta-données décrivant le contenu des sources de données. Ces dernières sont intégrées dans l'infrastructure en leur associant des facettes qui prennent en charge les différences techniques.

La notion de méta-données permet de résoudre certains problèmes liés à l'hétérogénéité et à la distribution. Les méta-données sont des données permettant de décrire d'autres données (avec généralement un plus haut niveau d'abstraction) facilitant leur réutilisation et l'interopérabilité des composants les manipulant.

A l'image des mécanismes d'interopérabilité que nous avons évoqués précédemment dans ce document (voir section III.4), les méta-données sont utilisées dans des domaines d'application spécifiques ou des approches de méta-modélisation (CDIF ou MOF [OMG 1997e]). [Busse et al. 1999] propose la classification suivante :

- les **méta-données techniques** décrivant les mécanismes techniques d'accès des composants, tels que les protocoles, la vitesse de connexion, le coût des requêtes, etc. Elles sont utilisées pour résoudre l'hétérogénéité technique et l'hétérogénéité des interfaces ;
- les **méta-données logiques** qui concernes les schémas et leurs relations logiques. Les méta-données logiques sont disponibles dans le dictionnaire des données dans les SGBD classiques ou dans les diagrammes de classes pour les SGBDOO ;
- les **méta-modèles** permettent de résoudre l'interopérabilité de plusieurs schémas issus de modèles de données différents (hétérogénéité des modèles de données) ;
- les **méta-données sémantiques** qui permettent de fournir une description sémantique des concepts manipulés par un SIF. Nous trouvons en particulier les travaux sur les ontologies ;
- les **méta-données relatives à la qualité** qui décrivent les propriétés des systèmes d'information en ce qui concerne leur qualité (robustesse, teneur aux pannes, fréquence des mises à jour, ergonomie, etc.). Ces descriptions sont utilisées à des fins d'optimisation ;
- les **méta-données concernant l'infrastructure** qui permettent aux utilisateurs de (re)trouver les informations pertinentes dont ils peuvent avoir besoin. On trouvera particulièrement les travaux portant sur les index et thésaurus, etc. (à ne pas confondre avec la description du contenu de ces derniers qui relèvent des méta-données sémantiques) ;
- les **méta-données relatives aux utilisateurs** permettant de décrire les responsabilités et les préférences, les profiles, etc. des utilisateurs.

Les méta-données interviennent donc à tous les niveaux (Présentation, Fédération, Fondation) de l'architecture d'un SIF.

III.5.1.7 Deux types de systèmes fédérés

Concernant le niveau fédération de l'architecture précédente, il peut revêtir deux formes selon la gestion du contrôle : serré ou lâche.

Fédération "serrée"

Une fédération "serrée" comporte un schéma global unifié constituant le schéma d'accès pour l'ensemble des utilisateurs. Au contraire, une fédération "lâche" ne comprend pas de schéma global mais offre simplement un langage de requêtes des données des composants.

Dans le cas de la fédération "serrée", le schéma global constitue un sous-ensemble de l'union des concepts importants des schémas des composants de la fédération. Il est donc indispensable que les correspondances soient définies entre les schémas des composants et le schéma global ; pour se faire, il est courant de recourir aux ontologies ou aux règles de correspondance (à partir de méta-données logiques parfois). Le processus de correspondance peut se faire de manière ad-hoc (par l'humain) ou automatique. Cependant, aucun système à ce jour n'a réussi dans ce domaine [Busse et al. 1999].

L'avantage de ce type de fédération pour l'utilisateur est que ce dernier n'a pas besoin de connaître le schémas de chaque composant mais seulement le schéma global. Cependant, l'utilisateur doit fournir un mécanisme de translation établissant les correspondances... Ces dernières étant bien souvent définies par un expert du domaine d'application du système fédéré.

Fédération "lâche"

Au contraire de la précédente, une fédération "lâche" n'a pas recours à un schéma unique global, mais fournit un langage de requêtes multi-bases de données (LRMBD[29]) qui est une abstraction des langages de requêtes des différents composants de la fédération et qui permet de cacher les problèmes d'hétérogénéité technique et l'hétérogénéité des langages. Cependant et dans ce cas, les utilisateurs ont la charge de résoudre ces problèmes d'hétérogénéité au niveau de tous les composants. Pour permettre de résoudre ce problème d'hétérogénéité "schématique" le langage "commun" doit couvrir un large panel de concepts et surtout d'éléments rencontrés au niveau des schémas... Ce qui n'est pas possible, par exemple, en utilisant le langage SQL.

Pour faciliter la perception des composants aux utilisateurs, il est commun de définir une vue "unique" pour chaque composant de la fédération; cette vue sera disponible aux utilisateurs qui pourront les utiliser comme s'il s'agissait de relations globales. Chacune de ces vues est formulée à l'aide du LRMDB.

III.5.1.8 Le modèle de données dans un SIF

Le niveau fédération d'un SIF est basé sur un modèle de données spécifique appelé le modèle de données canonique [Sheth et Larson 1990] ou modèle de données commun. Les schémas des fédérations de type "serré" sont des schémas de ce modèle. Dans le cas des fédérations de type "lâche", seul le langage utilisé pour accéder aux sources de données est basé sur ce modèle de données.

L'utilisation du modèle commun de données restreint considérablement le type de composants participant à la fédération, à cause des cas impossibles de conversions entre certains modèles. [Busse et al. 1999] présente quelques unes des incompatibilités entre les modèles de données les plus populaires. D'autres difficultés sont également rencontrées lorsque des modèles riches doivent être intégrés dans un modèle plus pauvre (par exemple, intégrer un modèle orienté-objet dans un modèle relationnel s'accompagne de pertes de connaissances sémantiques).

III.5.2 Les bases de données fédérées

En reprenant le cas des systèmes d'information fédérés que nous avons abordé, une base de données fédérée est formée de bases de données indépendantes [Saltor et al 1996]. Cette fédération est obtenue en considérant l'interopérabilité des différents systèmes de bases de données (appelés également composants).

L'interopérabilité et l'intégration de SGBD ont été largement étudiées. En particulier, de nombreuses études se sont intéressées à l'intégration de schémas hétérogènes.

Le processus d'intégration

L'intégration est définie dans ce domaine comme la combinaison, et l'interfaçage des données et des fonctions d'un système vers un ensemble cohérent [Heiler et al. 1991]. Le processus d'intégration comprend l'identification des relations et des dépendances entre les données et les procédures [Zisman et Kramer 1995]. De nombreux outils, méthodes et solutions pour intégrer des schémas de bases de données ont été proposés que l'on pourra trouver dans [Zisman et Kramer 1995]. En dépit des propositions et même de certains produits commerciaux (DATAPLEX, DDTS, PEGASUS, etc.) l'intervention humaine ne peut être évitée pour la résolution des conflits. Cette intervention est raisonnable lorsque le nombre de

[29] Le terme anglophone rencontré est "multidatabase query language (MDBQL)".

composants est raisonnable. En tout cas, pour un processus d'intégration, plusieurs résultats peuvent en découler.

[Batini et al. 1986] présentent une comparaison des méthodologies pour l'intégration de schémas et proposent les étapes importantes qu'il faut accomplir :

1. *Pré-intégration* : consiste à analyser l'ensemble des schémas avant leur intégration de façon à identifier quelle est la meilleure stratégie à utiliser. Cette stratégie dépend du choix des schémas à intégrer, l'ordre de cette intégration et s'il s'agit d'intégrer tout ou partie des schémas.
2. *Comparaison des schémas* : les schémas sont comparés et analysés de façon à déterminer les relations entre les différents concepts, les conflits possibles les propriétés inter-schémas.
3. *Conformité des schémas* : correspond à la résolution des conflits permettant de fusionner les différents schémas.
4. *Fusion et restructuration* : il s'agit de procéder à l'unification des schémas issus d'intégrations de schémas intermédiaires. Le schéma global résultant doit posséder les propriétés suivantes : (a) complétude signifiant que ce composant original doit contenir toutes les informations présentes avant le processus d'unification, (b) minimalisme, garantissant que les concepts ne sont présents qu'une seule fois, (c) compréhensibilité permettant aux utilisateurs de comprendre facilement le schéma unifié.

[Spaccapietra et al. 1992] divisent le processus d'intégration en deux phases. La première, nommée *phase d'investigation*, doit déterminer les points communs et les différences entre les schémas. La seconde phase concerne l'intégration effective, souvent semi-automatique.

Schéma global, architecture et interopérabilité

De nombreux prototypes et systèmes de base de données fédérés proposés sont basés sur l'architecture de référence à cinq niveaux proposée par [Sheth et Larson 1990] (voir Figure III.26).

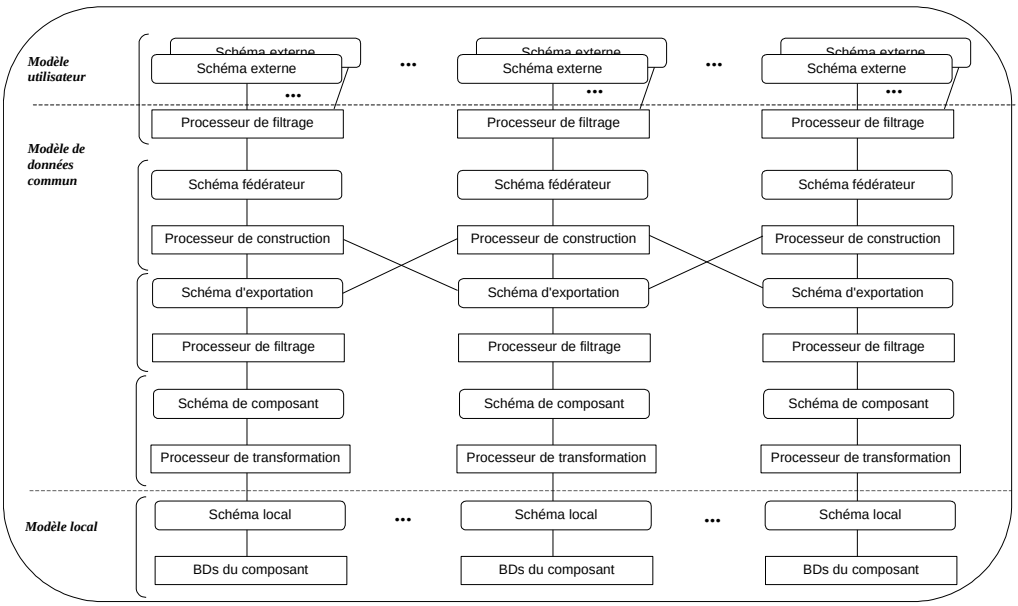

Figure III.26 Architecture du schéma à cinq niveaux (dans [Saltor et al. 1996] d'après [Sheth et Larson 1990])

Premièrement, un modèle canonique pour l'ensemble de la fédération doit être adopté. Les schémas locaux des composants sont alors transformés vers le modèle commun de données par le *processeur de transformation*, fournissant le *schéma du composant*. Ce schéma est ensuite filtré par le *processeur de filtrage* vers un ou plusieurs *schémas d'exportation*. A partir de ces derniers, un *schéma fédéré* est construit – ce processus est réalisé par le *processeur de construction* et s'appelle **l'intégration de schéma** ; plusieurs schémas fédérés peuvent exister pour une même fédération. Enfin, à partir d'un schéma fédéré, plusieurs *schémas externes* sont déduits par le *processeur de filtrage*, pour différents utilisateurs (ou catégories d'utilisateurs) du système de base de données fédéré. Nous avons donc :

> Le **schéma local** : il s'agit du schéma conceptuel d'un composant base de données exprimé dans le même modèle de données interne au composant; ainsi, plusieurs schémas locaux peuvent être exprimés dans différents modèles de données.
>
> Le **schéma de composant** : est le résultat de la transformation d'un schéma local vers un modèle commun de données qui a précédemment été choisi ; cette transformation est effectuée par le processeur de transformation.
>
> Le **schéma d'exportation** : représente le sous-ensemble d'un schéma de composant rendu disponible à la fédération et aux utilisateurs de cette dernière. Il faut noter que toutes les données d'un composant ne sont pas rendues disponibles. Le processeur de filtrage est utilisé pour fournir les contrôles d'accès et les opérations permises.
>
> Le **schéma fédéré** : il s'agit de l'intégration de plusieurs schémas d'exportation ; il inclut les informations concernant la distribution des données; ces informations sont générées lors de l'intégration des schémas d'exportation. Il est possible d'avoir plusieurs schémas fédérés à l'intérieur d'un même SBDF.
>
> Le **schéma externe** : il définit un schéma, pour un utilisateur ou pour une application ou pour une classe d'utilisateurs et d'applications. Un processeur de filtrage est utilisé pour analyser les commandes sur un schéma externe et pour garantir leur conformité avec le contrôle des accès et les contraintes d'intégrité du schéma fédéré.

Lorsqu'une fédération se forme ou lorsqu'un SGBD est sur le point de participer à une fédération existante, un processus de négociation est mis en place[30]. Chaque SGBD négocie quelles données parmi celles qu'il possède doivent être rendues accessibles (exportées) au reste de la fédération (à quelles catégories d'utilisateurs ou utilisateurs de la fédération) et quelles parties de ces données peuvent être lues ou/et mises à jour et par qui. Une fois que la fédération est constituée, une requête utilisant un schéma externe (correspondant à son schéma fédéré – par le processeur de filtrage) sera décomposée en sous-requêtes vers les composants de BD concernés (sollicitant le processeur de construction), qui seront transformées vers leur schéma local et soumises à leurs SGBD correspondants.

Comme nous l'avons vu dans le cas des SIF [Busse et al. 1999], un SGBDF peut être plus ou moins "lâche". Dans le cas "serré", la fédération met en place une administration de base de données au niveau fédération qui (1) se charge du processus de négociation avec les différentes administrations "locales" (celles des composants), (2) qui est responsable du processus d'intégration de schéma et (3) qui gère les schéma externe et fédéré. A l'opposé, une fédération "lâche" n'a pas d'administration fédérale (commune) ; ainsi, chaque utilisateur doit sélectionner quel composant accéder et il doit résoudre les problèmes d'hétérogénéité syntaxiques et sémantiques, incluant le processus d'intégration de schéma.

Bien que l'architecture à cinq niveaux soit assez générique pour pouvoir être adaptée à des cas particuliers, trois points ne sont cependant pas bien couverts par cette proposition :

[30] Un exemple de négociation peut être trouvé dans [Stonebraker et al. 1994a, Stonebraker et al. 1994b].

1. plusieurs composants (BD) issus de fédérations différentes ;
2. schémas externes exprimés dans des modèles utilisateurs différents d'un modèle canonique ;
3. différences sémantiques au niveau du schéma fédéré.

[Saltor et al. 1996] ont proposé, pour résoudre ces différents points, une architecture à huit niveaux que nous ne présenterons pas dans ce document.

La conclusion de ces travaux est qu'il n'est pas possible de considérer l'interopérabilité de différents SGBD sans étudier la similarité et l'équivalence des données concernées. Le problème de l'équivalence sémantique est crucial. Actuellement, la solution consiste à définir un modèle canonique approprié ; cette définition est un processus nécessaire mais laborieux qui ne peut être fait de manière automatique.

III.5.3 Les outils de workflow fédérés et systèmes coopératifs

Dans les années quatre vingt dix, un terme nouveau, "les systèmes d'information coopératifs (SIC)" est né, qualifiant la prochaine génération des systèmes d'information [Papazoglou et Schlageter 1998]. Le manifeste [De Michelis et al. 1998] trace un cadre de travail et d'étude pour les SIC, selon trois axes inter-liés :

1. *l'axe système* recouvrant l'information, les outils de workflow et d'autres systèmes logiciels offrant un support à certaines tâches. Cet axe recoupe les problèmes d'hétérogénéité et d'interopérabilité des systèmes concernés ;
2. *l'axe de la collaboration de groupes* qui recouvre comment les personnes travaillent ensemble sur un processus commun ou autres projets. Cet axe est aussi traité par la communauté de recherche portant sur les systèmes d'information collaboratifs et les collecticiels[31] ;
3. *l'axe organisationnel* qui recouvre l'organisation au sens général incluant la gestion des objectifs, le plan d'affaire, les stratégies, la gestion des projets et les flux résultant.

Ce découpage montre en fait que ce domaine ne s'occupe pas simplement de la gestion des échanges d'information mais également de la coordination des différents acteurs (personnes, systèmes, etc.). [De Michelis et al. 1998] montrent que la gestion des changements, la prise en compte de l'évolution sont deux dimension orthogonales à chacun de ces axes et sont deux des aspects importants des futures recherches.

Les architectures de ces systèmes sont basées sur les modèles que nous avons déjà présentés dans ce chapitre (courtiers, médiateurs, etc.).

En fait, la question de l'interopérabilité d'outils de workflow pour constituer une fédération a été étudiée sous le même angle que celui des bases de données fédérées que nous avons vues : les deux approches ont pour objectif l'intégration de composants hétérogènes à l'intérieur d'un système global (appelé *fédération*). La différence principale est que les différents composants d'une fédération de bases de données peuvent avoir différents modèles de données alors que les composants de workflow peuvent avoir différents modèles (conceptuels) de workflow [Geppert et al. 1998] ; il n'est pas toujours possible/souhaitable de considérer un modèle unique s'appliquant à l'ensemble des composants contrairement aux bases de données fédérées. Le cas d'un modèle unique partagé à été l'objet des approches WfMC [WfMC 1995, WfMC 1996a, WfMC 1996b]. D'autres en revanche, comme [Casati et al. 1996] prennent une optique inverse à savoir que les composants gardent leur autonomie et ne partagent pas de

[31] Le terme anglophone est Computer Supported Cooperative Work (CSCW).

modèle commun. La solution du modèle commun est irréaliste dans le cas d'entreprises distribuées et autonomes [Geppert et al. 1998, Tombros et Geppert 2000]. On en déduit qu'une fédération de systèmes de workflow doit s'affranchir des contraintes suivantes :

- aucune supposition sur l'existence d'un modèle homogène et global ;
- les spécifications d'un système de workflow restent privées à ce système ;
- l'exécution d'un processus de workflow reste sous le contrôle de chacun des systèmes de workflow.

Différentes architectures possibles

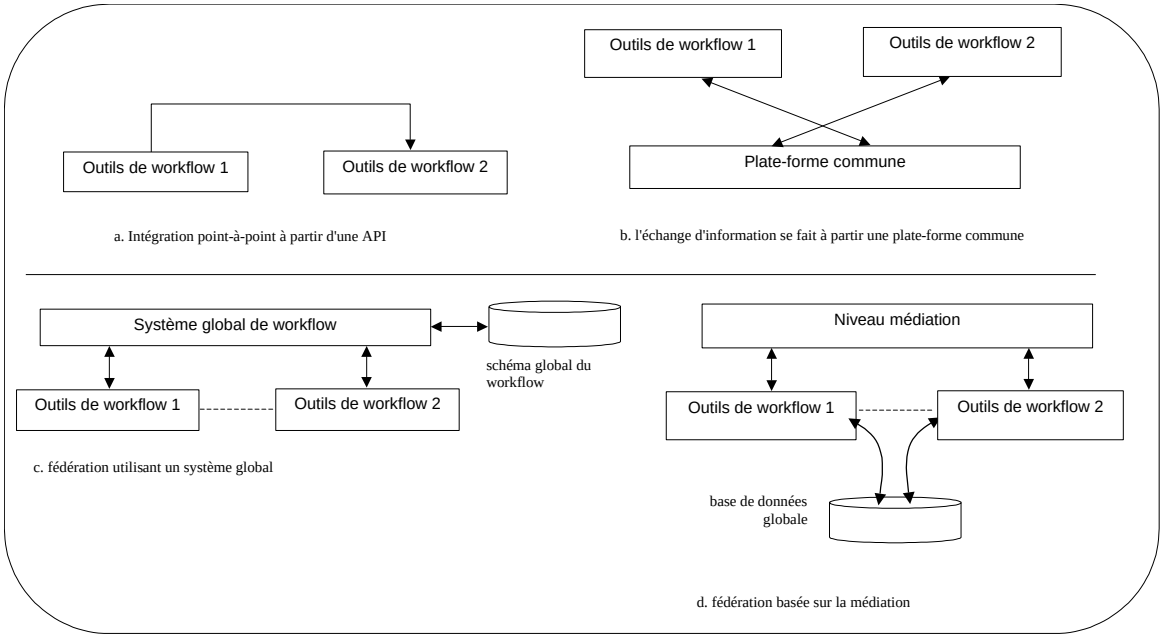

Figure III.27 Différents choix d'architecture pour les fédérations de workflow

Dans le premier cas (Figure III.27.a.), les systèmes de workflow interagissent entre eux directement via leurs API. Le principal problème de ce mode de fonctionnement est qu'ils doivent se connaître mutuellement (et donc ils doivent implémenter autant d'interfaces que de composants, ce qui est très pénalisant en terme de maintenance et d'évolution).

Le second cas (Figure III.27.b.)est basé sur l'intégration de données entre différents systèmes de workflow: les systèmes interagissent via le SGBD (voir Wide [Casati et al. 1996]). Cette configuration ne se prête pas à la problématique des fédérations de composants distribués pour laquelle les entreprises sont réfractaires à autoriser les accès à leurs données. Ce mode n'est donc pas adapté aux fédérations inter-entreprises.

Dans le troisième cas (Figure III.27.c.), il existe un niveau d'intégration global tel qu'il est décrit dans [Sheth et Larson 1990] pour le cas des fédérations de bases de données ; cette couche est un système de workflow à part entière. Les spécifications de ce dernier sont transcrites en types/concepts de chaque composant local. Le fait est que la plupart des entreprises disposent déjà de leurs propres outils de workflow et que, dans ce cas, il faut leur installer le système de la couche intermédiaire.

Le dernier cas (Figure III.27.d.)mentionne une fédération construite sur une couche de médiation permettant à chaque système/composant d'invoquer les autres composants. Contrairement au cas précédent, cette couche intermédiaire n'est pas elle-même un système de workflow complet.

La spécification globale des systèmes de workflow est telle qu'un flux (workflow) *wf* est appelé global si au moins un de ses sous-flux est exécuté par un système de workflow différent du système exécutant *wf*. Pour pouvoir spécifier les workflow globaux il faut :

1. que les différents systèmes puissent connaître quels sont les types/concepts offerts par les autres systèmes ;
2. que les types d'un système puissent être représentés par les autres systèmes ;
3. que les composants logiciels soient représentés comme des entités de calcul et que l'allocation des tâches puisse être spécifiée.

La première condition est satisfaite s'il existe une base de donnée pour la fédération contenant l'ensemble des informations portant sur les types exportés par les différents systèmes de workflow de la fédération. Pour chacun de ces types, les informations suivantes leurs sont associées :

- son nom ;
- un ensemble de paramètres typés d'entrée ;
- un ensemble de paramètres typés de sortie ;
- un ensemble de systèmes de workflow qui implémentent et qui exportent ce type ;
- un ensemble de systèmes de workflow qui importent ce type.

Pour pouvoir satisfaire le second point, un système de workflow doit pouvoir importer les types qui sont définis dans la base de données commune.

Modéliser la fédération revient donc à définir un processus de workflow global selon la connaissance (incomplète) que l'on possède des différents systèmes de workflow, participants de la fédération. Les invocations des différents systèmes sont alors réalisés par la couche de médiation et les conversations sont opérés par les systèmes eux-mêmes.

Les composants de médiation constituent la "colle" entre les systèmes de workflow locaux [Geppert et al. 1998]. Ces composants sont de deux types :

- du côté du client (c'est-à-dire où le processus global de workflow s'exécute), d'autres composants de workflow appelés entités de calcul[32] qui fournissent les fragments de workflow importés sous la forme de services ;
- du côté du serveur (c'est-à-dire les composants de workflow distants où des sous-processus du processus global de workflow peuvent être exécutés) ou des clients sont les médiateurs entre les clients et les systèmes locaux de workflow.

Ainsi, pour exécuter un fragment de workflow distant (auprès d'un système de workflow distant SW1), l'entité de calcul communique avec le client approprié (le client du système de workflow SW1). Ce dernier peut automatiquement démarrer le processus via l'interface cliente de son système de workflow et, une fois le processus terminé, reçoit les résultats et les retourne à l'entité de calcul à l'origine de l'invocation (qui elle-même retourne le résultat au système de workflow initial).

Les technologies utilisées pour implémenter cette architecture reposent sur CORBA-OMG [Miller et al. 1996, Geppert et al. 1998] et les composants sont des objets distants.

[32] Nous trouverons le terme anglophone "processing entity".

III.5.4 Evaluation

Nous remarquons que les travaux sur les systèmes fédérés diffèrent peu des travaux portant sur les bases de données fédérées et possèdent un certain nombre de leurs caractéristiques (du à l'importance et à la place centrale des sources de données dans les systèmes d'information). Ces travaux ont permis de considérer les systèmes fédérés avec les trois dimensions que sont l'autonomie des participants, la distribution et l'hétérogénéité. D'autre part, les points suivants ont été particulièrement abordés :

- formes, modèles et formalismes canoniques ;
- technologies permettant aux différents participants d'une fédération d'interopérer (nous retrouvons en particulier, les bus logiciels, les technologies supportant les applications réparties, etc. – OMG-CORBA, etc.).

Particulièrement, les types d'architectures ("serrée" ou "lâche"), dans les bases de données notamment, sont une déclinaison de la notion de contrôle (un contrôle rigide ou alors lâche).

En revanche, dans le domaine des bases de données fédérées, les composants sont, par nature des composants dont leur univers de discours respectif varie peu (il s'agit toujours de composants de base de données). En ce sens, les solutions proposées ne permettent de résoudre qu'un sous ensemble de notre problématique (à savoir les formes canoniques et les moyens de les obtenir en partant des composants vers le schéma commun et vice versa). Pour autant, il n'existe pas de solution automatique à ce jour.

Nous noterons également que les propositions ont surtout considéré les problèmes d'intégration de schéma, d'interopérabilité sous les aspects syntaxiques et sémantiques. En revanche, la question considérant la flexibilité du contrôle (et de l'autonomie) des composants n'a pas été une des priorités. Les approches bases de données et workflow ont fait l'hypothèse que l'autonomie des composants devait être garantie sans en mesurer le degré.

Les approches dans les domaines des processus (de workflow, logiciel, …) doivent en plus, traiter les problèmes de recouvrements, de cohérence entre les états locaux des participants et l'état global de la fédération. Cette notion d'état global ne fait d'ailleurs pas l'objet de consensus à ce jour.

La section suivante va dresser un bilan de l'ensemble des travaux abordés au cours de ce chapitre pour voir, en quoi certains points restent ouverts et en quoi la problématique que nous exposons dans cette thèse est toujours un problème de recherche. Nous verrons ensuite les solutions qui peuvent se dégager, pouvant servir de base à notre proposition en vue de résoudre tels ou tels aspects de notre problématique.

III.6 Bilan et conclusion

III.6.1 Les critères d'évaluation

Nous remarquons que des différences importantes existent entre les différentes propositions et, entre autre :

1. les approches **intrusives** (qui imposent des interventions au niveau des composants en vue de leur assemblage – il faut modifier les composants) ;
2. les approches **non intrusives** dont :

a) celles qui font des hypothèses *à priori* sur les capacités ou les spécifications que doivent respecter les composants ;

b) celles qui ne font aucune hypothèse *à priori*.

Une position orthogonale viserait à classer les propositions d'assemblage selon leur approche :

i. **déductive** (on déduit, à partir des techniques d'assemblage, des caractéristiques sur les composants) ;

ii. **inductive** (on déduit les techniques d'assemblage à partir des caractéristiques des composants à assembler).

Des travaux qui ont été présentés au cours de ce chapitre, nous retenons les aspects suivants qui ont été abordés :

- la spécification et la modélisation des composants,
- la préservation de l'autonomie des composants,
- le type d'approche adoptée (intrusive ou non-intrusive, déductive ou inductive),
- l'interopérabilité (syntaxique et sémantique),
- la flexibilité du contrôle,
- les paradigmes d'assemblage,
- la diversité des moyens techniques,
- la modélisation globale du système.

Ces critères peuvent être répertoriés selon qu'ils portent sur les composants, sur les moyens et méthodes permettant l'assemblage ou sur le système global.

Critères **Domaines**	**CBSE**	**EAI**	**SMA**	**Interopérabilité**	**SIF**
modélisation des composants	oui	non	faible	non	faible
autonomie des composants	non	oui	non	non	faible
approche intrusive ou non intrusive	intrusive	non intrusive	intrusive	intrusive et non-intrusive	intrusive et non intrusive
approche déductive ou inductive	déductive	inductive	déductive	inductive	déductive
interopérabilité syntaxique			oui	oui	oui
interopérabilité sémantique			faible	oui	oui
flexibilité du contrôle	non	non	non	non	non
paradigmes d'assemblage	non	utilise	oui	oui	non
moyens techniques	oui	oui	peu	non	oui
modélisation globale	arch. logicielle	non	non	non	non

Tableau III.2 Bilan récapitulatif suivant les domaines

Le tableau (Tableau III.1), présente une évaluation des différentes approches suivant les domaines face aux critères que nous avons sélectionnés.

III.6.2 Conclusion

Notre problématique se situe selon une approche non intrusive, déductive et dont l'objectif est la modélisation globale de fédérations qui est le cas le plus général et pour lequel peu de propositions sont à recenser. Les travaux que nous avons regardés se répartissent selon les autres cas (voir tableau III.2). Nous notons que les approches CBSE (ingénierie des systèmes basés sur des composants) ou EAI (intégration d'application d'entreprise) n'apportent pas de réponse concrète aux problèmes d'assemblage de composants existants (notamment de type COTS) : elles n'en mesurent que les effets, relatent les différentes alternatives et adoptent une approche pragmatique (bas niveau). De leurs côtés, les bases de données fédérées ne traitent le cas que de composants de "type" SGBD… Et non n'importe quel composant (COTS ou autre).

Macroscopiquement nous pouvons constatées que les propositions qui ont porté sur l'assemblage des composants n'ont pas permis d'étudier les composants et leurs spécificités. A l'inverse, les propositions de modèles et les spécifications de composants n'ont pas fourni de méthodes d'assemblage de haut niveau ou alors, s'accompagne d'un haut niveau de complexité (CCM). Le problème de l'interopérabilité couplé à celui de l'hétérogénéité des composants reste encore une question de recherche.

Cette question est en toile de fond des systèmes fédérés tels que nous les avons étudiés. Par contre, les solutions apportées sont généralement figées à la fois dans la composition des systèmes (ajout ou retrait de composants), dans leurs modes de fonctionnement (dans le sens où les paradigmes d'interaction ne sont pas modifiables) et dans la prise en compte de l'évolution.

En particulier, nous constatons les limitations suivantes dans les systèmes fédérés et les approches d'intégration :

1. les composants issus du marché sont peu pris en compte (les composants souvent considérés sont des composants développés *a posteriori*, en respectant les spécificités de l'environnement auquel ils doivent s'intégrer) ;
2. les modes de fonctionnement, de contrôle, les paradigmes d'interaction sont figés et ne peuvent être modifiés "facilement" : chaque système propose un paradigme particulier qui ne peut être modifié. La flexibilité n'est donc pas permise ;
3. pas de modélisation explicite de la fédération (modèle de composants, modèle de fonctionnement, etc.). Les concepts concernant la modélisation des fédérations n'ont pas été exhibés.

En regard des travaux dans le domaine de l'EAI, nous ciblons le niveau 4 proposé par [Schmidt 2000], c'est-à-dire fournir des concepts permettant la modélisation de solutions complètes basées sur des applications existantes, garantissant un certain nombre de propriétés (transactions, persistance, etc.).

Ces questions encore ouvertes constituent les objectifs de nos travaux dans le cadre de cette thèse. Le chapitre suivant va porter sur la présentation de notre proposition concernant ce que nous appellerons par la suite, des *fédérations d'outils* (logiciels).

Chapitre IV CONCEPTION ET CONSTRUCTION DE FEDERATIONS D'OUTILS…

Conception et Construction de Fédérations d'Outils...

Après avoir présenté le domaine des processus logiciels et leurs supports – les EGLCP – de première génération (deuxième chapitre), le chapitre précédent a présenté des travaux dans des domaines divers permettant à la fois de caractériser les constituants (participants) de la fédération, d'avoir un aperçu sur les différentes approches et technologies permettant d'assembler les constituants et, pour certains systèmes, de fédérer ces derniers sous certaines hypothèses (domaines d'application, spécifications, etc.).

Ce quatrième chapitre a pour but de présenter lnos travaux sur les fédérations d'outils. Après une introduction, rappelant nos objectifs et le contexte, nous présenterons successivement les problèmes qui se posent dans le contexte des fédérations d'outils et, pour chacun d'entre eux, nous montrerons comment l'approche que nous proposons constitue une avancée dans ce domaine. Nous aborderons la constitution d'une fédération d'outils (section IV.2), la gestion de son contrôle (section IV.3), l'adaptation des outils leur permettant de participer aux fédérations (section IV.4) ainsi que l'architecture globale (section IV.5). Nous illustrerons nos propos par des exemples tirés de scénarios.

IV.1 Problématique

IV.1.1 Rappel des besoins et des objectifs

Au fur et à mesure du document, plusieurs besoins ont été identifiés. Nous rappelons la problématique que nous étudions à savoir permettre de construire une fédération dont les participants sont des outils logiciels, certains d'entre eux étant disponibles sur le marché, ou issus de systèmes patrimoines, ou quelconques.

L'enjeu est de permettre de construire une application logicielle à partir d'applications logicielles existantes dont le fonctionnement sera cohérent pour l'utilisateur final, permettant de satisfaire les buts qu'il s'est fixé.

Nous nous intéressons à l'ensemble des outils (moyens informatiques) qui constituent ce que nous avons qualifié de *fédération d'outils* ainsi qu'aux moyens permettant l'assemblage de ces outils.

Cette fédération d'outils se place dans un contexte :

1. de **distribution** : les outils sont potentiellement répartis sur plusieurs sites géographiquement éloignés ;
2. **d'hétérogénéité** : les outils sont hétérogènes ;
3. **d'accessibilité** : les outils doivent pouvoir être sollicités ;

4. **d'autonomie** : les outils doivent participer à la fédération mais doivent pouvoir, si le besoin s'en fait sentir, rester disponibles dans leurs environnements respectifs (autonomie locale) de même que l'information qu'ils gèrent localement peut être à caractère confidentiel ;
5. **d'évolution** : les outils peuvent être sollicités différemment. Ils peuvent quitter la fédération, ou y participer à n'importe quel moment selon les exigences et les besoins du processus ;
6. de **coopération** : les outils sont amenés à coopérer de manière à atteindre les buts de la fédération ;
7. de **contrôle** : le mode de fonctionnement de la fédération (la manière de gérer la coopération des outils) doit pouvoir être contrôlé et être flexible.

IV.1.2 Qu'entend-on par outil ?

Notre objectif est de construire des fédérations dont les participants sont :

- des composants développés à la demande,
- des applications ou systèmes patrimoines,
- des composants issus du marché (COTS).

Le cas des systèmes à base de composants développés à la demande, parmi lesquels nous trouvons les fédérations, a été abordé dans le domaine de l'ingénierie des systèmes à base de composants dont certains travaux ont été présentés dans le troisième chapitre de ce document.

Comme nous l'avons montré dans le troisième chapitre, le cas des COTS est le plus restrictif sachant qu'un outil de type COTS :

- est une boîte noire dans le sens où son code source n'est pas disponible et ne peut donc pas être modifié,
- est conçu pour s'auto-suffire, c'est-à-dire qu'il n'a pas besoin d'autres outils pour fonctionner (il a besoin d'un environnement minimal[33]),
- interagit avec son environnement de différentes façons (utilise le système de fichiers, interagit avec l'utilisateur, etc.),
- peut avoir un comportement non déterministe et non prédictif dans le contexte de la fédération du fait de l'utilisateur.

Nous utiliserons, dans la suite du document, le terme outils pour qualifier un composant issu du marché (COTS) ou toute application logicielle existante, pouvant être de granularité variable.

IV.1.3 Qu'est-ce qu'une fédération d'outils ?

De la définition donnée par [Robert 1989], nous pouvons extraire trois idées fondamentales :

i. l'identification des entités indépendantes ;
ii. l'assemblage des entités pour satisfaire un but ;
iii. l'autorité commune.

[33] S'exprimant notamment en terme de ressources.

> Aussi, nous définissons une *fédération d'outils* logiciels comme étant un assemblage d'outils logiciels et autonomes, soumis à une autorité commune en vue de réaliser une application.

Cette définition permet de distinguer :

- l'identification du but ;
- l'identification des outils pouvant satisfaire le but (réaliser l'application) ;
- l'assemblage des outils soumis à une autorité commune.

Il s'agit de définir les dimensions de base de tout système à base de composants, c'est-à-dire [Bolusset et al. 1999] :

i. le *quoi* (but) ;
ii. le *qui* (les outils) ;
iii. le *comment* (l'assemblage).

Les fédérations font partie de ces systèmes à base de composants avec, en plus, les caractéristiques suivantes :

- l'indépendances des participants (outils) ;
- l'autorité commune.

D'un point de vue informel, les outils vont coopérer pour satisfaire le but de la fédération à laquelle ils appartiennent. La notion de coopération (et celle de fédération aussi) fait que les outils ont, entre eux, un espace de partage, rassemblant éventuellement des moyens, des connaissances leur permettant de coopérer (la notion de fédération sous-entend une mise en commun de "moyens").

D'un point de vue opérationnel, nous souhaitons contrôler la coopération des participants, donnant plus ou moins d'importance aux "libertés" de chacun d'eux.

D'un point de vue conceptuel et technique, les outils hétérogènes doivent être adaptés (voir les travaux de [Abts et Boehm 1997, Hayden et al. 1999, Stonebraker 1999, SEI 1997] sur l'adaptation) pour permettre le "partage" ou une certaine compréhension de la "communauté" et qu'ils puissent se joindre à la fédération.

Nous avons donc à répondre aux questions suivantes :

- Comment pouvons-nous exprimer la "communauté" ?
- Comment pouvons-nous contrôler la fédération ?
- Comment pouvons-nous faire en sorte que les outils participent à la communauté qui aura été définie ?

Ces trois questions constituent une partie du plan de ce chapitre et seront abordées au cours des sections suivantes.

IV.1.4 Une fédération d'outils : les problèmes à résoudre

En partant des caractéristiques des outils présentés précédemment et en regard de nos objectifs, les fédérations d'outils s'accompagnent de problèmes qu'il faut résoudre :

1. le recouvrement (partiel) des fonctionnalités et des concepts des outils de la fédération,
2. l'indéterminisme des comportements de chacun des outils dans le cadre d'une fédération,
3. les incohérences entre les concepts tels qu'ils sont définis et leurs implémentations au niveau des outils,
4. les différents niveaux d'abstraction.

Nous allons, dans les sections suivantes, présenter notre proposition et montrer en quoi elle propose des solutions pour les quatre problèmes que nous venons d'énoncer. Pour ce faire, nous prendrons deux scénarios nous permettant, tout au long de ce chapitre, d'illustrer nos propos.

IV.1.5 Présentation des scénarios

Nous définissons deux scénarios qui se prêtent à l'étude des fédérations d'outils. Le premier scénario se place dans le domaine des processus logiciels alors que le second porte sur un système de transport impliquant plusieurs entreprises.

Processus logiciel : cas du transfert de produit

Comme nous l'avons vu dans le deuxième chapitre, la modélisation des processus logiciels a fait l'objet de nombreux travaux. Chaque EGLCP propose une façon de modéliser le processus, à partir d'un méta-modèle, ce dernier reposant sur des concepts. Bien qu'il y ait des différences notables d'un méta-modèle à l'autre, souvent selon l'approche adoptée, les concepts de base proposés dans [Promoter 1999] se retrouvent fréquemment utilisés. Par exemple, l'environnement APEL [Estublier et al. 1998a] utilise, entre autres, les concepts d'activité, de produits, etc. [Amiour et Estublier 1998].

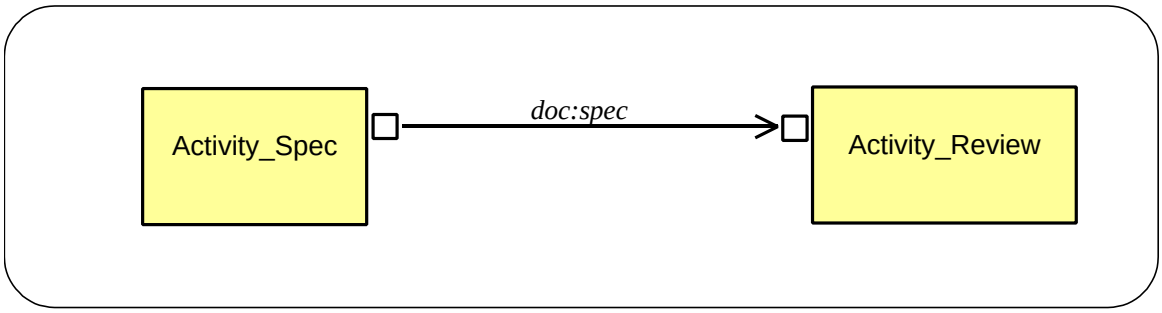

Figure IV.1 Transfert de produit entre deux activités

Le cas du scénario de transfert de produit se présente sous la forme d'un processus. Ce dernier comporte deux activités avec, entre elles, un flux de données (voir Figure IV.1). A partir de ce modèle, plusieurs instances (du modèle) peuvent ainsi être définies, chacune pour un processus spécifique. La définition et l'état courant de chacune de ces instances lors de l'exécution, sont stockés dans une base de données (selon un processus de réification [Amiour 1999]). Les instances vont être exécutées par le moteur de processus (voir second chapitre).

L'environnement APEL dispose de moyens permettant la manipulation des objets de la base de données : ce sont des opérations (méthodes en langage java).

La figure IV.1 décrit un processus avec une activité de spécification et une activité de revue avec, entre elles, le transfert d'un document de spécification. Selon le scénario, lors de

l'exécution, la fin de l'activité de spécification provoque une opération de transfert du document vers l'activité de revue.

Système de transport : gestion des tickets de transport

Le second scénario porte sur la gestion de tickets pour un système de transport. Plusieurs entreprises de transport (bus, trains, transports publics, aéronautique, etc.) décident de créer et de participer à une fédération dont l'objectif est de permettre, pour un utilisateur (un usager), de prendre plusieurs moyens de transports différents (chacun d'entre eux étant géré par une des entreprises de la fédération) au cours d'un itinéraire avec le même ticket (voir Figure IV.2).

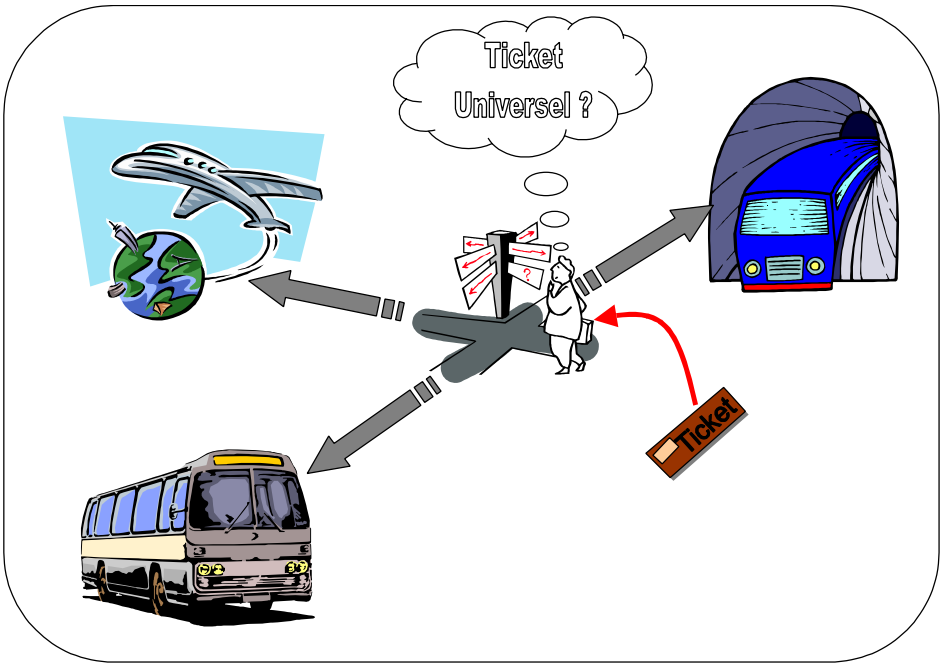

Figure IV.2 Gestion des tickets de transport

Nous allons présenter dans la suite du document, les solutions que nous proposons face aux problèmes qui ont été identifiés avec, pour illustrations, les deux scénarios cités précédemment.

IV.2 Constitution d'une fédération : l'idée communautaire

L'idée de base est de considérer que l'existence d'une fédération ne se justifie que par le fait qu'il existe une application à réaliser ; l'application est à la fois le besoin (le cahier des charges) et à la fois le but (de la fédération).

Nous pensons que les outils, pour pouvoir participer à une fédération, doivent pouvoir être fédérés. Sachant que l'application est le centre de la fédération, nous pensons que les artefacts de l'application représentent le point de convergence, l'espace partagé (tout ou partie) des outils.

IV.2.1 Le domaine d'application

Comme nous l'avons écrit plus haut, le scénario de transfert de produit entre les deux activités se situe dans le domaine des processus logiciels. Majoritairement, les outils de ce domaine sont ce que nous appelons des systèmes sensibles aux processus (PSS[34]). Ces outils partagent les mêmes concepts même s'il existe des différences notables entre eux, particulièrement liées à l'approche sur laquelle ils reposent (approche orientée but, approche orientée activités/produits, etc.). Cependant, les domaines des processus (logiciel, workflow), etc. impliquent généralement différentes catégories d'outils : les éditeurs, les compilateurs, les gestionnaires d'espaces de travail, les gestionnaires de configuration, les gestionnaires de révisions, etc. Ces outils peuvent être géographiquement distribués.

Imaginons une fédération comprenant deux gestionnaires de versions, RCS et CVS, ainsi qu'un gestionnaire de configuration Adele qui sera utilisé pour mettre à jour les espaces de travail des différents utilisateurs.

Dans le cas du transfert de produit (ce produit étant un document de spécifications), chacun de ces outil doit "connaître" le document à transférer mais chacun décline sa propre "vision" du document en fonction de son univers de discours.

Dans le cas de la gestion du ticket de transport, chacun des outils/système impliqués dans la fédération va devoir "connaître" le ticket comme entité à part entière mais chacun de ces systèmes va en avoir une vision particulière en fonction également de son univers de discours et de son domaine d'application spécifique ; par exemple, les concepts permettant de définir un ticket de métro diffèrent sensiblement de ceux d'un ticket d'avion, de train, etc. Par exemple, le ticket d'avion mentionne une zone fumeur ou non-fumeur, un numéro de place, la classe du voyageur, etc. contrairement au ticket de métro.

Ainsi, les outils sont impliqués dans différentes "sections", "aspects" du ticket pour le cas du ticket de transport (respectivement du document dans le cas du scénario de transfert).

IV.2.2 Un univers commun : recouvrement de concepts

Nous proposons de définir un ensemble de concepts communs et selon un haut niveau d'abstraction, permettant de mettre sous un même édifice l'ensemble des connaissances communes et des concepts manipulés par les outils composant la fédération. Ces concepts "communs" sont directement liés à l'application de la fédération. Ainsi, pour ce qui concerne les deux scénarios, aurons-nous à définir une abstraction du document de spécification, du ticket de transport...

> Nous définissons un **univers commun** (UC) comme étant un ensemble d'entités, chacune étant partagée par au moins deux outils participant à la fédération. Une entité est une instance d'un concept. Notons que lorsqu'un concept de l'application ne concerne et n'intéresse qu'un seul outil de la fédération, il n'est nullement besoin qu'il apparaisse dans l'univers commun (ce dernier regroupant ce qui est partagé).

Chaque outil décline sa propre vision de tout ou partie des concepts de l'UC qui peut recouvrir (totalement ou partiellement) d'autres visions d'autres outils de la fédération. Il est important de noter que les outils ne partagent pas tous forcément l'ensemble des concepts de l'UC. L'UC représente donc une abstraction du monde "réel", c'est-à-dire de l'application. Logiquement et en reprenant les deux scénarios, les objets locaux des outils, implémentant les entités abstraites ainsi que les actions des outils sur ces objets sont les actions du monde "réel" (ce qui est réellement fait par les outils en dehors de l'univers commun, lors de l'exécution de la

[34] Nous retenons l'abréviation anglophone PSS (Process Sensitive System).

fédération). Ces actions concernent les entités locales de chaque outil (puisque les outils n'ont pas connaissance de l'application – autonomie et auto-suffisance).

L'univers commun est donc le lieu où l'information commune est stockée dans un format permettant aux différents outils de se synchroniser :

- synchronisation de l'état local de l'outil en fonction de l'état de l'UC,
- synchronisation de l'état (commun) de l'UC en fonction des états locaux des outils.

L'UC contient donc les entités communes (instances des concepts communs) et l'état de chacune de ces entités lors de l'exécution. Lorsqu'une entité de l'UC change d'état, des notifications vont être envoyées aux outils concernés par ce changement d'état ; ils seront alors "libres" de synchroniser leur état local en fonction de l'état commun. Inversement, lorsque l'état local d'un outil est modifié[35] et que cette modification concerne une entité de l'UC, l'état de ce dernier doit être modifié à son tour.

Nous remarquons que le fonctionnement même d'une telle fédération, basée sur l'état commun [Heimbigner 1992] ne concerne pas uniquement la synchronisation des outils et la cohérence de leurs univers locaux en fonction de l'univers commun (et vice versa), mais concerne également la façon d'effectuer le contrôle de la fédération. Ainsi, les outils sont "libres" de s'abonner aux changements d'état de l'UC, sont "libres" de recevoir des notifications, sont "libres" de modifier l'état de l'UC... sans aucune contrainte.

> L'univers commun possède les caractéristiques suivantes [Estublier et al. 2001a] :
>
> - il est *abstrait* (voir ci-dessus) ;
> - il est *incomplet (non exhaustif)*. L'univers commun ne contient que les données strictement nécessaires à l'exécution "correcte" de la fédération : ce qui est partagé ;
> - il est *dynamique*. L'univers commun contient l'état courant de l'application, cet état évoluant au cours du temps.

Nous pouvons donc dire qu'il n'y a pas de contrôle global de la fédération, que cette dernière suit un mode de fonctionnement "anarchique" [Estublier et Verjus 1999] dans le sens où l'initiative appartient uniquement aux outils et à l'exécution de ces derniers (c'est le code de l'outil, sa logique fonctionnelle interne qui va décider du comportement de l'outil). En ce sens, la cohérence de l'état de l'UC n'est assurée que si les outils "s'entendent" sur le processus de synchronisation (absence de règles de contrôle).

Le scénario du transfert de produit

Imaginons qu'un utilisateur, acteur du processus ouvre le document de spécifications avec un éditeur, apporte quelques modifications au document puis enregistre le document ainsi modifié. Le gestionnaire des espaces de travail va mettre à jour l'état de l'UC concernant l'objet de l'UC représentant le document (des changements d'état de l'objet : "édité", "sauvegardé", etc.) alors que d'autres outils vont réagir lors de l'enregistrement du document (l'outil CVS va faire une révision du document ainsi modifié).

Le scénario du ticket de transport

Imaginons qu'un client décide d'annuler le voyage qu'il avait prévu, le voyage ayant fait l'objet de plusieurs réservations auprès des différentes entreprises de transport impliquées dans une portion de l'itinéraire. Cette annulation va se traduire par la suppression de l'objet représentant

[35] Souvenons-nous que les outils interagissent avec leur environnement.

le ticket dans l'univers commun. L'ensemble des outils gérant chacun une portion de l'itinéraire vont devoir être informés de l'annulation pour annuler (localement) chaque portion.

Dans les deux scénarios, les outils agissent (peuvent modifier l'état de l'UC) et réagissent (aux changements de l'état de l'UC) sans se connaître mutuellement [Estublier et al. 2001a].

IV.2.3 Définition d'une fondation

Ainsi, les outils participent à l'évolution de l'univers commun en agissant sur son état. Dans le cadre que nous venons de décrire, nous appelons [Estublier et al. 2001b] :

> **fondation** : l'univers commun et l'ensemble des outils contribuant[36] à l'évolution de cet univers commun, impliqués et caractérisant une application.
>
> Selon cette vision, l'état (abstrait) de l'application à un instant donné est l'état de l'univers commun appelé **univers commun de l'application** (UCA), à ce même instant.
>
> L'état de l'univers commun est modifié par sollicitations d'un ensemble de services (synchrones ou asynchrones) constituant les **services de la fondation**.
>
> La **fondation de l'application** (FA) est l'UCA et l'ensemble des outils impliqués dans l'évolution de l'UCA.

Nous définissons donc une fondation *Fond.* comme :

$$Fond. = \{UC_{Fond}, O_1, O_2, ..., O_n, MI_{Fond}\}$$

avec :

- UC_{Fond} l'univers commun de la fondation,
- $(O_1, O_2, ...O_n)$ les outils de la fondation,
- MI_{Fond} les mécanismes permettant l'interaction des outils avec l'univers commun.

Cela se traduit, au niveau de l'architecture d'une fondation par un composant appelé "univers commun" permettant le stockage des entités et de leurs concepts, une interface (API) permettant la manipulation de ces entités (consultation et modification de l'état de l'univers commun) ainsi qu'un gestionnaire de notifications. Le principe de fonctionnement de l'univers commun a été largement étudié [Heimbigner 1992, Estublier et al. 1998b, Ben-Shaul et Kaiser 1995, Ben-Shaul et Kaiser 1998, Estublier et Verjus 1999, Estublier et al. 2001].

Ainsi, dans le cas des scénarios, nous pouvons exhiber deux fondations de l'application, chacune correspondant à un des scénarios.

> La première est constituée d'un univers commun contenant l'état du processus en cours (états des instances définies à partir d'un méta-modèle de processus – dans notre cas il s'agit du méta-modèle d'APEL [Dami 1999]), les outils participant à l'évolution de cet univers commun étant des outils des EGLCP, des EGLCP ou des PSS, et une infrastructure de communication (CORBA, Java/RMI, JMS, etc.).
>
> La seconde est constituée d'un univers commun contenant l'état des entités de l'application (en l'occurrence, les tickets, les itinéraires et l'ensemble des informations nécessaires à la gestion des tickets et des itinéraires), les outils de l'application de

[36] L'acte de contribution consiste, comme nous l'avons vu, en une action ou une réaction (via souscription).

gestion des tickets (les systèmes effectuant la gestion des tickets), et une infrastructure de communication (CORBA, Java/RMI, JMS, etc.).

Paradigme de fonctionnement de la fondation

Dans les deux cas, chacune des fédérations peut être définie comme une fondation ; cette fondation est rattachée à l'application la caractérisant :

- le processus comprenant les deux activités est *l'application* du premier cas ;
- l'application de gestion de tickets de transport est *l'application* du second cas.

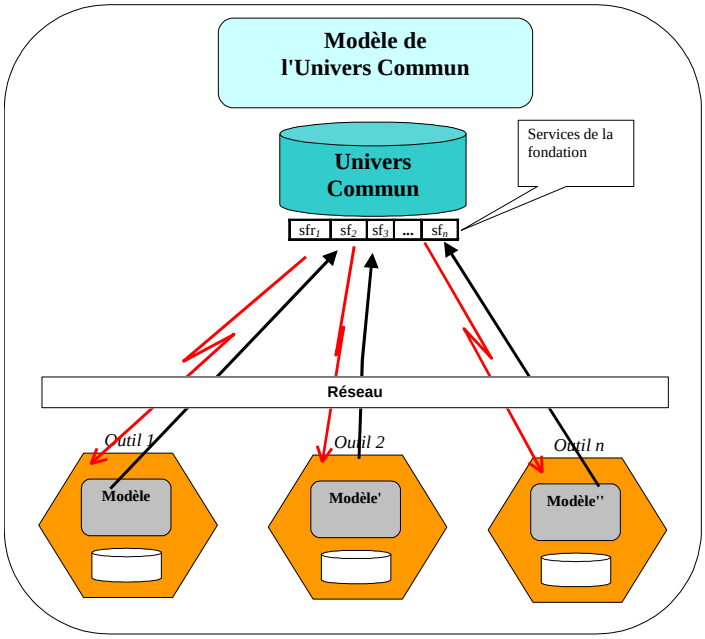

Figure IV.3 Le paradigme de l'anarchie [Estublier et al. 2001]

Le fonctionnement de la fédération est caractérisé par le fonctionnement de la fondation (identité entre les deux). Cette dernière, comme nous l'avons vu, suit un comportement qualifié d' "anarchique" [Estublier et Verjus 1999, Estublier et al. 2001a] (voir figure IV.3).

> Les services de la fondation sont appelés les services de la fédération ou, **services fédéraux**.

Selon ce modèle, la connaissance "commune" à l'ensemble des outils et permettant leur synchronisation est "librement" gérée. Il n'y a aucun contrôle de cohérence de la fédération (absence de règle de fonctionnement explicite).

IV.3 La gestion du contrôle d'une fédération d'outils

Cette section va porter sur les approches et les mécanismes permettant de contrôler les fédérations d'outils. En effet, comme nous venons de le voir, les fédérations précédentes (appelées aussi *fondations de l'application*) n'intègrent aucun mécanisme de contrôle de leur cohérence. Le contrôle permet essentiellement de répondre aux problèmes :

- de recouvrement de concepts entre les outils ;

- d'indéterminisme de la fédération (comportement non déterministe des outils du fait des utilisateurs).

Nous verrons que la gestion du contrôle permet également d'appliquer des paradigmes de fonctionnement différents. La gestion du contrôle que nous proposons, qui vient en réponse aux problèmes identifiés, est répartie selon trois dimensions complémentaires :

1) le contrôle des initiatives des outils dans le cadre d'une fédération ;
2) le contrôle de l'évolution de l'état courant de l'univers commun ;
3) la coordination des outils.

IV.3.1 Pourquoi le comportement des outils doit-il être contrôlé ?

Les outils participant aux fédérations que nous étudions, en particulier les COTS, sont des outils bien souvent déterministes intrinsèquement : cela veut dire qu'ils ont, par l'intermédiaire de leur code source, un fonctionnement "programmable" et déterminé. En revanche, la plupart des outils ont des interactions avec un environnement. Cet environnement est constitué de l'ordinateur sur lequel s'exécute l'outil (comprenant entre autre, un système de fichiers, des périphériques, etc.), et des utilisateurs sollicitant l'outil de différentes manières :

- au travers d'une interface utilisateur,
- en utilisant un langage de commandes (si l'outil peut être invoqué par des commandes),
- etc.

Ces interactions s'effectuent dans les deux sens, (1) de l'outil vers l'environnement (c'est le cas, par exemple, lorsqu'un éditeur va enregistrer un fichier : il modifie le système de fichier), (2) de l'environnement vers l'outil (cas des sollicitations des utilisateurs par exemple).

Aussi, le comportement intrinsèquement déterministe de l'outil n'est plus garanti lorsque ce dernier est immergé dans un environnement fertile en sollicitations.

L'indéterminisme lié à la fédération

En plus de l'indéterminisme propre à chaque outil (résultant de leurs environnements respectifs), nous identifions un niveau d'indéterminisme propre à la fédération. En effet, dans le cadre des fédérations, les outils peuvent se comporter de façon aléatoire et de façon non prédictive. Cela est la conséquence de leur propre "vision" et l'interprétation de l'état courant de l'univers commun d'une part, et du fait qu'ils peuvent faire preuve d'initiatives (voir section IV.3.2) d'autre part. Le résultat de cette situation est qu'ils ont la possibilité de modifier à tout instant, sans concertation, l'état de l'univers commun (par modification des entités de cet univers commun).

Dans le cas de chacun des deux scénarios, des comportements non contrôlés peuvent produire un état indésirable de l'univers commun. Par exemple dans le scénario du ticket de transport, le fait que sur suppression du ticket (entité abstraite de l'UC) les différents outils aient la liberté de supprimer, à leur niveau, la portion de l'itinéraire les concernant rend le comportement global de la fédération indéterministe. Par exemple, imaginons que les suppressions des différents itinéraires doivent être ordonnées (pour des raisons de dépendances [Sichman 1995, Allouche 1998], de séquence, etc. entre les différentes portions de l'itinéraire) ; dans le mode "anarchique" ou "anarchique contrôlé" les outils vont être avertis par notifications mais vont pouvoir réagir librement et individuellement sans tenir

compte d'un comportement global *souhaitable* (ordonnancement, état global, etc.) de la fédération.

Il s'agit donc, dans le cas général de pouvoir définir, lorsque cela s'avère nécessaire[37], une politique de coordination des différents outils de la fédération face à un changement (ou une volonté de changement) de l'univers commun.

IV.3.2 Contrôler les initiatives des outils

En reprenant les deux scénarios, issus de domaines d'application distincts, nous constatons que les outils de chacune des fédérations présentent des similitudes à la fois dans certains des concepts comme nous l'avons vu mais également dans leurs fonctionnalités. Il est possible que plusieurs outils fassent la même chose simultanément (par exemple, l'édition d'un même fichier, la réservation d'un même itinéraire par deux systèmes – et donc deux entreprises de transports – différents, etc.) et ainsi, modifient une même entité de l'univers commun simultanément ce qui aurait pour conséquence de produire une incohérence au niveau de l'état de ce dernier. La "libre entreprise" et l'initiative laissée aux outils est source d'incohérence de l'univers commun.

Il serait donc intéressant de pouvoir disposer de mécanismes permettant de contrôler les initiatives des outils, en limitant par exemple leurs possibilités, en dédiant un "aspect" de l'UC à un outil spécifique, etc.

En partant d'une fondation de l'application, cela reviendrait à contrôler [Estublier et al. 2001a] :

1. les actions des outils sur l'UC,
2. les réactions potentielles des outils face aux changements d'état de l'UC.

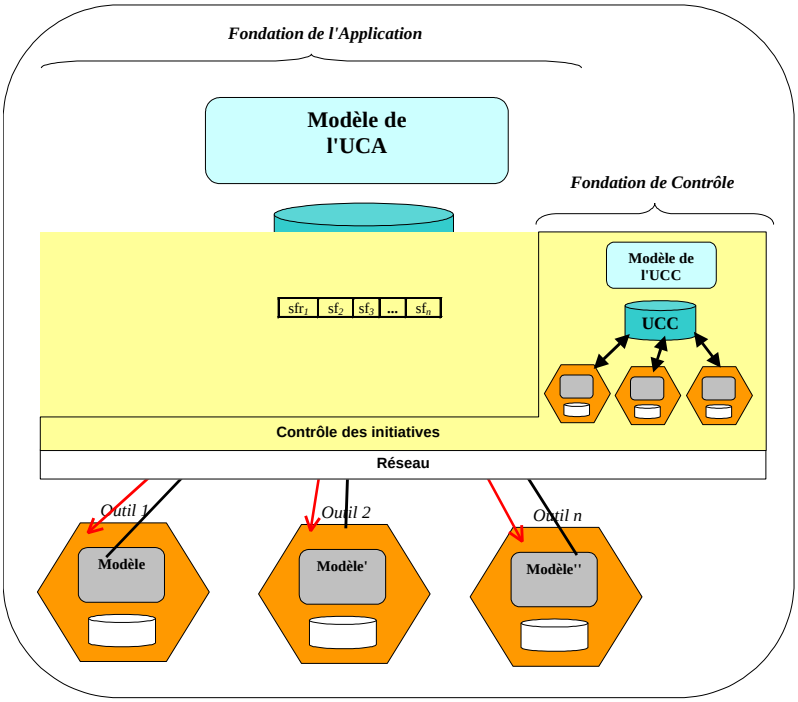

Figure IV.4 Le paradigme de l'anarchie contrôlée [Estublier et al. 2001]

[37] Laissé à l'appréciation du concepteur de la fédération.

Nous proposons de placer un niveau de contrôle (voir Figure IV.4) entre les outils et l'univers commun dont l'objectif est de :

1. contrôler les invocations de l'UC émanant des outils,
2. contrôler les souscriptions des outils et les notifications vers les outils,

c'est-à-dire, de contrôler les initiatives des outils dans le cadre de la fédération. Ainsi, le concepteur de la fédération va pouvoir définir des interdictions concernant les outils. Une interdiction associe un service fédéral et un outil en qualifiant s'il s'agit d'une invocation (fait d'invoquer le service fédéral en question) ou d'une observation (fait de recevoir des notifications lorsque le service fédéral en question a été invoqué).

Le but est de donner la possibilité, au concepteur de la fédération, de restreindre le comportement des outils de la fédération en inhibant tout ou partie de leurs initiatives (inhibition des "libertés").

Pour permettre de mettre en oeuvre les interdictions il nous faut un composant de contrôle dont le but est d'intercepter et de contrôler la légitimité des invocations et des notifications.

IV.3.2.1 Univers de contrôle et modèle de fédération

Par analogie avec la définition de la fondation de l'application (voir section IV.2.3), nous définissons une **fondation de contrôle** (FC) qui respecte la définition générale, à savoir :

- un univers commun, appelé ici **univers commun de contrôle** (UCC), contenant pour ce niveau, un ensemble d'interdictions[38]. Ces interdictions concernent les outils de la fédération et portent sur les invocations ou les notifications liées aux services fédéraux ;
- un ensemble d'outils impliqués dans cet univers commun de contrôle, mettant en oeuvre les mécanismes de contrôle des initiatives émanant des outils. Ces outils, lors de l'exécution, interprètent le modèle de contrôle (contenant entre autre les interdictions) et effectuent les opérations d'interception, de filtrage, éventuellement, de blocage ;
- des mécanismes d'interactions permettant aux outils de la FC de communiquer avec l'UCC[39].

Selon le schéma proposé (voir Figure IV.4), les fédérations ainsi définies sont composées de deux univers communs :

1. l'UCA dédié à l'application de la fédération (qui concerne les outils de la fédération – les outils de l'application),
2. l'UCC dédié au contrôle de la fédération (qui concerne les outils de contrôle uniquement).

L'univers commun de contrôle contient l'ensemble des concepts, des entités basées sur ces concepts permettant de définir le comportement global de la fédération et le comportement de chacun des outils de la fédération.

La fondation de contrôle comprend différents niveaux (ces niveaux seront introduits progressivement au cours de ce chapitre), dont le niveau de contrôle des initiatives. Les

[38] Nous verrons dans les sections suivantes que l'univers commun de contrôle contient bien plus que la définition des interdictions.
[39] Ces interactions peuvent être de simples appels de services basés sur le mécanisme RPC si elles s'effectuent entre deux outils (logiquement) différents et (éventuellement) distants.

concepts utilisés pour la définition de chacun des niveaux sont les concepts de l'univers commun de contrôle (couvrant ainsi l'ensemble des niveaux).

Le niveau de contrôle des initiatives

Les composants de la fondation de contrôle sont chargés de la mise en oeuvre de la politique de contrôle. Pour ce niveau, cela correspond à appliquer l'ensemble des interdictions ce qui se traduit par :

- intercepter les invocations des outils de l'application vers l'UCA, les demandes de souscriptions des outils de l'application, les notifications en direction des outils de l'application,
- vérifier si aucune interdiction ne concerne, soit les outils et le sujet de l'invocation (le service de l'UCA invoqué), soit les demandes de souscription, soit les notifications,
- laisser passer les invocations, les demandes de souscriptions, les notifications si aucune interdiction n'a été définie,
- ou de les bloquer sinon.

Ainsi, nous définissons une application pour les composants de la fondation de contrôle :

> l'*application* de la fondation de contrôle n'est autre que le *contrôle* de la fondation de l'application.

IV.3.2.2 Le contrôle des initiatives et la gestion du recouvrement

Le fait de donner la possibilité de contrôler les initiatives (donc potentiellement de les inhiber) répond par la même occasion au problème du recouvrement entre les outils de la fédération. La gestion des recouvrements passe par l'allocation des "responsabilités" à chaque outil, cette dernière étant assurée par la fondation de contrôle. Nous allons voir comment le recouvrement entre les outils d'une fédération est géré dans les deux scénarios.

Le scénario du transfert de produit

Imaginons que lorsque l'activité de spécification est terminée, nous voulons que l'outil CVS fasse une révision de ce document mais que les autres gestionnaires de révision (RCS, etc.) ne fassent pas de même (car cela ne les concerne pas !). Il est alors utile de bloquer les demandes de souscriptions émanant des gestionnaires de révision (sauf CVS) concernant les notifications du transfert de produit et de bloquer les notifications dans le cas où cette interdiction surviendrait au cours de l'exécution de la fédération.

Seul l'outil CVS sera informé (par notification) du transfert de produit et pourra (il est "libre" au sens où l'initiative lui est laissée par réaction à la notification[40]) effectuer une révision du document.

Le scénario du ticket de transport

Dans le cas de l'annulation du ticket de transport, suivant la configuration de la fédération, il se peut que l'annulation soit sous la responsabilité d'un seul outil de la fédération. Dans ce cas, l'invocation de l'UCA permettant de supprimer un ticket ne sera disponible que pour l'outil en question. Les autres outils pourront alors souscrire à la suppression du ticket et mettre leurs états locaux à jour (suppression de la portion de l'itinéraire les concernant) en fonction de l'état de l'UCA.

[40] En ce sens, nous distinguons un comportement actif d'un comportement réactif.

La couche de contrôle des initiatives [Estublier et al. 2001a, Estublier et al. 2001b] permet donc de raffiner le mode de fonctionnement "anarchique" tel qu'il était proposé [Estublier et al. 1998b]. Nous allons aborder dans la section suivante, le problème de l'indéterminisme résultant de la liberté d'initiative laissée aux outils de la fédération.

IV.3.3 Contrôler l'évolution de l'univers commun

Le domaine des processus (et de leurs supports) – processus manufacturiers, logiciels, organisationnels, etc. - a abordé le cas de l'indéterminisme, résultant en partie de l'imprévisibilité des comportements humains utilisant les technologies qui leur sont mises à disposition.

En particulier et comme nous l'avons présenté dans le second chapitre portant sur les technologies dans le domaine des processus logiciel, un modèle de processus décrit le comportement attendu du monde réel [Cîmpan 2000] et l'évolution attendue de ce dernier [Estublier et al. 2001b]. Un moteur de processus interprète une (ou plusieurs) instance du modèle (voir section II.2.3.1) et met à jour les entités de l'UC (qui sont des abstractions des artefacts du monde réel). Les entités de l'UC sont décrites à l'aide de concepts décrivant une ontologie[41].

Des travaux récents ont porté sur l'étude des déviations entre le comportement attendu (ou modélisé), le comportement réellement observé [Cîmpan 2000] et les mécanismes de réconciliation [Alloui 1998] dans un environnement complet [Alloui et al. 1999].

Nous proposons d'utiliser les apports de ces technologies. En ce sens, nous pouvons définir un modèle de processus qui décrit le comportement attendu de l'univers commun de l'application (UCA) et l'évolution attendue de ce dernier.

> Ce modèle ainsi défini peut être vu comme l'expression formelle du **processus** permettant d'atteindre les buts de la fédération, c'est-à-dire, permettant de réaliser l'application.

[41] Dans notre cas, il s'agit d'une ontologie propre aux processus logiciels.

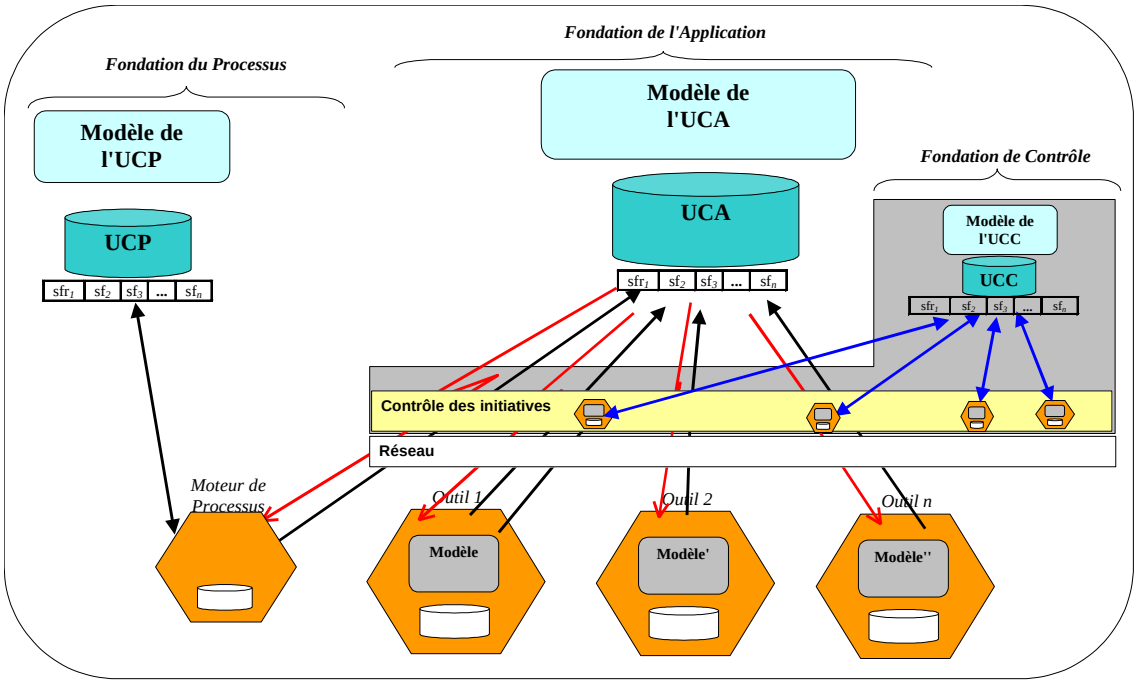

Figure IV.5 Apports des technologies processus pour gérer l'indéterminisme

Nous identifions une troisième fondation (voir Figure IV.5) appelée **fondation du processus** (FP). Par analogie avec la définition générale d'une fondation (voir section IV.2.3), cette fondation du processus comprend :

- un univers commun, appelé ici **univers commun du processus** (UCP), contenant un modèle de processus ainsi qu'une ou plusieurs instances avec leurs états respectifs, basées sur un méta-modèle dont les concepts appartiennent au domaine des processus[42],
- un ensemble d'outils impliqués dans cet univers commun de processus. Ces outils sont les outils identifiés dans les technologies processus, à savoir les composants "classiques" d'un EGLCP, des PSS, etc., entre autres les composants participant (lors de l'exécution) au processus d'évolution de l'UCA (outils de monitoring, outils de stratégie d'évolution, outils de décisions pour la réconciliation, outil d'implémentation des changements, etc. voir [Cunin et al. 1999, Cunin 2000, Alloui et al. 2000]),
- des mécanismes d'interactions permettant aux outils de la fondation du processus de communiquer avec son univers commun (l'UCP).

Ainsi, la fondation du processus est un environnement centré processus pouvant être plus ou moins complet (donc intégrant plus ou moins d'outils) suivant les besoins[43].

Selon le schéma proposé (voir Figure IV.5), les fédérations ainsi définies sont composées de trois univers communs :

1. l'UCA dédié à l'application de la fédération,
2. l'UCC dédié au contrôle de la fédération,
3. l'UCP dédié au processus de la fédération.

[42] Dans notre cas, le méta-modèle utilisé est le méta-modèle d'APEL [Dami 1999].
[43] L'environnement utilisé dans le cadre de nos travaux est l'environnement APEL [Estublier et al. 1998a, Amiour 1999, Dami 1999].

C'est le moteur de processus qui va se charger de l'exécution du processus (par interprétation du modèle instancié – voir section II.2.3.1) et va agir sur l'UCA pour lui appliquer le processus d'évolution défini et stocké (réifié[44]) dans l'UCP.

Par exemple, dans le scénario de gestion du ticket de transport, il est possible de définir un modèle (de processus) décrivant le processus de réservation d'un itinéraire avec les différentes vérifications et la création du ticket de transport. Ce modèle de processus appartient à l'UCP. A l'exécution, le moteur du processus va interpréter une instance du modèle et l'état de l'UCA (contenant l'entité abstraite "ticket") va être modifié par le moteur du processus. En effet, lors de l'édition définitive du ticket (pouvant être considérée comme une activité du processus), l'entité "ticket" de l'UCA va passer à l'état "édité". Ce changement d'état de l'UCA va potentiellement provoquer des actions/réactions des outils de la fondation de l'application (mise à jour de leur propre état local), etc.

La fondation du processus induit, en quelque sorte, une partie du comportement (l'évolution) de l'univers commun de l'application de la fédération.

Remarque : l'outil moteur de processus a une position singulière dans la fédération (composée ici de trois fondations). En effet, il fait partie de la fondation du processus mais il interagit également avec l'UCA. Or, ces deux univers communs peuvent reposer sur des ontologies bien différentes (cas général) basées sur des concepts n'ayant rien en commun. Nous constatons que dans l'exemple du scénario de gestion du ticket de transport, le ticket est à la fois un concept de l'UCA mais a également une représentation (abstraite) dans l'UCP (par exemple, en utilisant les concepts utilisés dans la méta-modélisation des processus logiciels, le ticket peut être considéré comme un *produit* – concept de l'UCP).

IV.3.4 Coordination des outils

Comme nous l'avons vu précédemment, le mode de fonctionnement "anarchique" n'inclut aucune règle de contrôle. L'indéterminisme et l'imprévisibilité sont deux caractéristiques inhérentes à ce paradigme. Le contrôle par le processus tel que nous venons de le proposer offre une solution à l'imprévisibilité dans le sens où il permet de définir le comportement *attendu* de l'univers commun de l'application (de la fédération). Cependant, il n'offre aucune solution à l'indéterminisme de la fédération et à un contrôle plus rigoureux en intervenant directement au niveau des outils. En effet, le contrôle par le processus agit sur l'UCA et non sur les outils.

Si nous reprenons le scénario du transfert de produit, le transfert du document entre les deux activités devrait impliquer plusieurs outils :

- le gestionnaire des espaces de travail qui va mettre à jour les espaces de travail des utilisateurs concernés (ceux de l'activité source et ceux de l'activité avale),
- les gestionnaires de révisions qui vont être amenés à faire une révision du document,
- etc.

Dans nombre de ces cas (où plusieurs outils sont impliqués dans le changement d'état de l'UCA), il est nécessaire que les actions des différents outils soient coordonnées pour garantir la cohérence du système. D'autre part, d'autres propriétés sont souvent à prendre en compte comme la gestion des transactions (faire en sorte que l'enchaînement de ces actions soient inclus dans une transaction), etc.

[44] Voir [Amiour 1999].

Par exemple, dans le scénario de gestion du ticket de transport, supposons qu'un client décide de changer son trajet par la réorganisation de l'enchaînement des portions de l'itinéraire. Dans le mode de fonctionnement "anarchique", les outils qui sont concernés par les différentes portions vont être notifiés de la réorganisation et vont modifier leurs états locaux en fonction du changement. Ce changement sera soumis à des vérifications locales propres à chacun des outils. Dans ce cas, localement, les changements peuvent être validés ou refusés. Ainsi, dans le cas où un changement local ne peut s'opérer (par exemple, il n'y a plus de place disponible dans le moyen de transport pour les dates après modifications), la réorganisation générale de l'itinéraire doit être annulée. Cela ne peut se faire par simple notification.

Nous définissons un niveau de contrôle de cohérence et de coordination (voir Figure IV.6) dont l'objectif est de permettre [Estublier et al. 2001b] :

- le contrôle et la gestion des transactions,
- le contrôle et la gestion des contraintes temporelles,
- le contrôle et la gestion de la coordination des actions des outils.

Ce niveau de contrôle (comme le montre la figure IV.6, ce niveau de contrôle fait partie de la FC) prend en charge tout ou partie des requêtes/invocations destinées à l'UCA et détermine quels sont les outils impliqués dans la gestion de ces requêtes, de ces invocations et exécute le comportement défini par le concepteur de la fédération (transaction, ordonnancement des actions des outils, etc.) par invocation des outils impliqués. Pour ce faire, ce niveau interprète un modèle de coordination (non détaillé ici). Le modèle de coordination est stocké dans l'univers commun de contrôle (UCC) ; ce dernier contient, comme nous l'avons vu, en plus du modèle de coordination, le modèle décrivant les outils de la fédération ainsi que le modèle de contrôle des initiatives individuelles.

Selon cette perspective, les outils peuvent être contrôlés plus finement dans le sens où la fondation de contrôle peut invoquer directement les services de ces derniers, en fonction du modèle de coordination. Dans le cas où les outils sont invoqués par la fondation de contrôle, ils n'ont plus aucune liberté (sollicitation synchrone, point à point).

Une des principales caractéristiques des niveaux de contrôle (individuelles et coordination) est leur parfaite transparence au niveau des outils : ces derniers n'ont aucune connaissance des différentes contraintes. L'un des rôles de la fondation de contrôle est justement d'intercepter les requêtes en destination de l'UCA (seul connu des outils) et d'exécuter le modèle de coordination à l'insu des outils.

Techniquement, les appels distants en direction de l'UCA vont être capturés ; pour ce faire, les composants de la fondation de contrôle agissent au niveau de l'infrastructure de communication permettant la re-direction des appels et des invocations. Nous détaillerons les aspects techniques et liés à l'implémentation dans le cinquième chapitre.

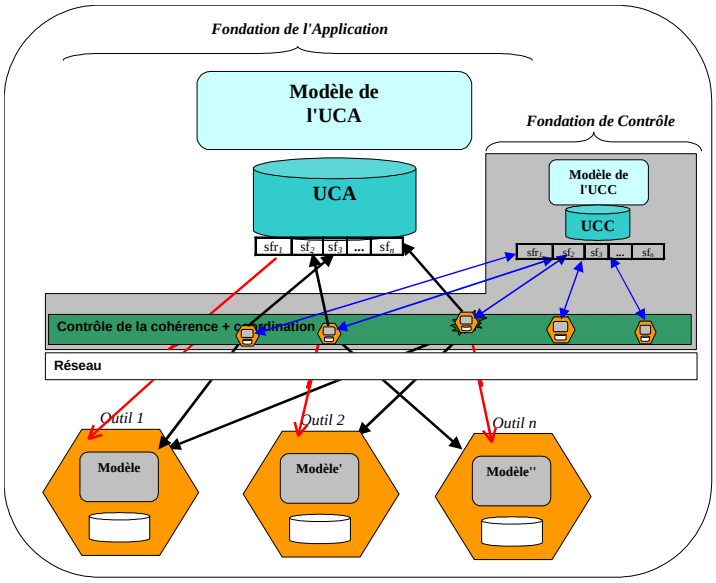

Figure IV.6 Contrôle et coordination

Un modèle de fédération basé exclusivement sur ce type de contrôle (excluant les précédentes alternatives de fonctionnement) suit le paradigme de la "dictature" [Estublier et al. 1998b, Estublier et Verjus 1999]. Dans ce cas, les outils de la fédération n'ont aucun droit (ils ne peuvent invoquer les services fédéraux). La fondation de contrôle prend tout en compte : elle invoque les outils directement et invoque les services fédéraux selon le modèle de coordination.

Le modèle de coordination que nous aborderons plus loin dans ce document comprend pour chaque outil, la définition des rôles joués, leurs droits et leurs devoirs dans le cadre des fédérations (chaque fédération particulière pouvant disposer d'un modèle de coordination particulier).

IV.3.5 Des fédérations multi-paradigmes

Par analogie avec une société humaine, le niveau de cohérence et de coordination joue le rôle d'un gouvernement imposant le comportement des individus pour un certain nombre d'aspects de la société. En effet, nous verrons que le concepteur de la fédération devra définir quels sont les *aspects* de l'application qui devront être contrôlés. Ceci permettra d'adapter (dosage du contrôle) le mode de fonctionnement de la fédération. Dans la réalité en effet, certains des aspects de l'application devront être fortement contrôlés alors que d'autres pourront être gérés en laissant l'initiative aux outils.

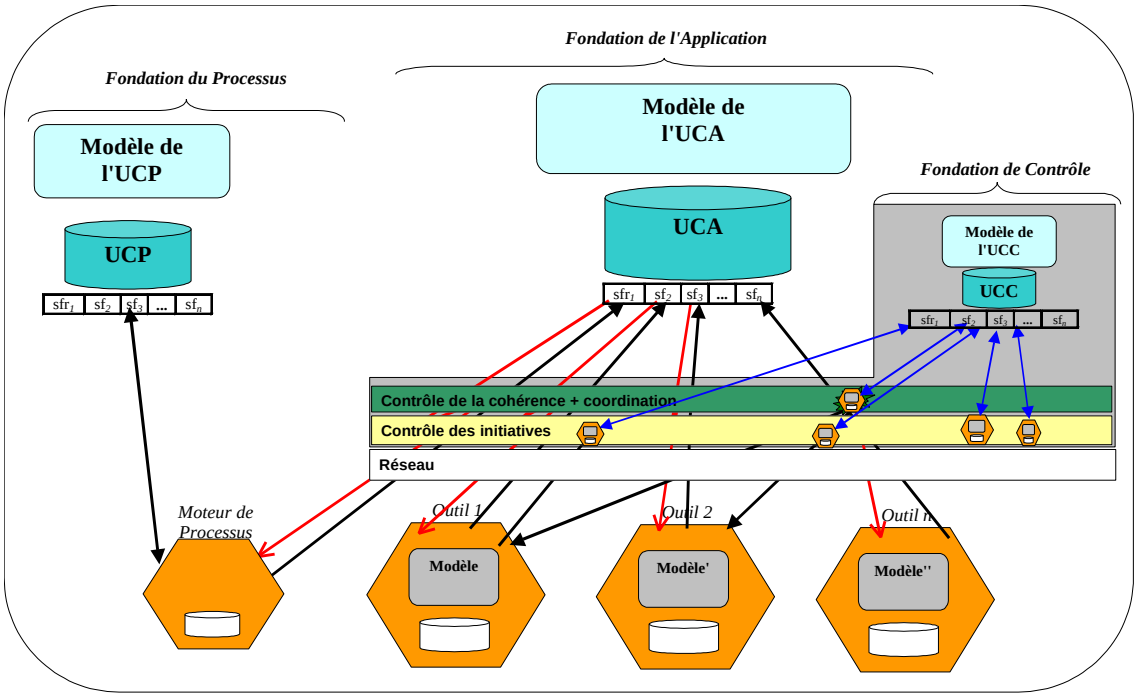

Figure IV.7 Contrôle selon plusieurs paradigmes [Estublier et al. 2001b]

Dans le scénario de gestion du ticket de transport, la suppression du ticket pourra être un aspect de l'application dont la gestion sera laissée à l'initiative des outils (style anarchique) alors que la modification de l'itinéraire sera un aspect de l'application dont la gestion sera fortement contrôlée (style dictatorial). A l'image d'une société, les fédérations peuvent implémenter des politiques mixtes, c'est-à-dire basées sur un mélange de paradigmes.

IV.3.6 Les concepts utilisés pour le contrôle des fédérations d'outils

Nous avons proposé essentiellement deux niveaux pour la fondation de contrôle, en charge du contrôle des fédérations d'outils :

- niveau de contrôle concernant les initiatives (invocations des services fédéraux et souscriptions aux services fédéraux) ;
- niveau de coordination et de cohérence.

Chacun de ces deux niveaux dispose de prérogatives de contrôle. La gestion des interdictions est à la charge du niveau de contrôle des initiatives et la gestion de la coordination incombe au niveau en charge de la coordination.

Pour matérialiser ces deux niveaux, nous introduisons les concepts suivants :

- le concept de **rôle**. Un rôle est une vision abstraite d'un outil : ce n'est pas un outil existant mais un outil défini avec les concepts de l'application (de la fédération). La description du rôle comprend :
  - les obligations, c'est-à-dire l'ensemble des services fournis et les souscriptions ;
  - les interdictions portant, soit sur les invocations des services fédéraux, soit sur les notifications résultant de l'invocation des services fédéraux ;
- le concept **d'aspect**. Chaque aspect contient l'expression de la coordination des différents outils, exécutée lorsqu'un service fédéral est invoqué. Par rapport à la

définition de la fédération donnée dans [Robert 1989], les aspects sont l'expression de l'*autorité commune* de la fédération.

La définition des rôles et des aspects, pour une fédération donnée, traduit le mode de fonctionnement de la fédération d'outils (sa "politique") et détermine le comportement de chacun des outils dans ce cadre. Ces concepts font partie des concepts de l'*univers commun de contrôle* de la fédération. Les mécanismes de contrôle et leur implémentation feront l'objet du chapitre suivant.

IV.3.7 Les possibilités de contrôle en réponse aux problèmes posés

Les différentes possibilités de contrôle d'une fédération d'outils permettent de répondre aux problèmes évoqués au début de ce chapitre à savoir :

- le recouvrement des fonctionnalités et des concepts des outils de la fédération est géré essentiellement par le niveau de contrôle des initiatives (section IV.3.2) ;
- les problèmes liés aux comportements non déterministes des outils et à l'évolution de l'univers commun de l'application sont traités :
    - par le niveau de coordination des outils (section IV.3.4) ;
    - par l'ajout d'une fondation du processus dont le rôle est de décrire puis d'exécuter le processus d'évolution de l'UCA (section IV.3.3).

La gestion du contrôle de la fédération permet de répondre aux problèmes de recouvrement, d'imprévisibilité et d'indéterminisme. Nous avons identifié deux autres problèmes [Estublier et al. 2001a, Estublier et al. 2001b] :

- l'incohérence entre les concepts "communs" de la fédération et leurs implémentations au niveau des outils de la fédération ;
- pouvoir rendre disponibles les outils afin qu'ils puissent intégrer une fédération d'outils.

Ces deux points vont faire l'objet de la section suivante. Ils montrent que des outils issus du marché ne peuvent pas être immergés dans un environnement coopératif (cas des fédérations) et être immédiatement opérationnels (pouvoir être sollicités). Au contraire, [Brown et Wallnau 1996, SEI 1997, Kontio 1996, Abts et Boehm 1997, Morisio et al. 2000] ont montré qu'une phase d'adaptation est nécessaire.

IV.4 L'adaptation des outils en vue de leur participation aux fédérations

Dans le cadre des fédérations d'outils telles que nous les considérons, les outils doivent pouvoir :

- être sollicités par la fédération pour qu'ils remplissent les tâches incombant aux rôles endossés ;
- solliciter les services fédéraux lorsque cela s'avère nécessaire (entre autre lorsque la modification de l'état local d'un outil affecte l'état de l'univers commun – cohérence).

Il faut donc fournir des moyens pour que, d'une part les services des outils soient rendus "disponibles" à la fédération (à la fondation de contrôle) et que, d'autre part des

correspondances puissent être définies entre l'ontologie de l'univers commun et l'ontologie de chacun des outils (et vice-versa) ; ainsi est-il nécessaire d'adapter les outils.

Notre problématique s'intéresse aux fédérations d'outils selon une approche "boîte noire". Cela veut dire que, pour bon nombre des outils et en particulier pour les outils de type COTS, il n'est pas possible de les modifier (approche non-intrusive). Aussi, pour permettre leur intégration dans les fédérations d'outils, nous définissons deux niveaux d'intégration (donc d'adaptation) :

1. l'intégration conceptuelle ;
2. l'intégration technique.

Dans le premier cas, nous introduisons le concept de **représentant** alors que dans le second, nous présenterons les **façades** des outils.

IV.4.1 L'adaptation conceptuelle

Nous allons voir qu'intégrer des outils hétérogènes est source d'incohérence. L'adaptation conceptuelle a pour but de résoudre ces incohérences par l'association, à chaque participant de la fédération, d'un **représentant**.

IV.4.1.1 Les raisons de l'incohérence

Les solutions que nous avons proposées jusqu'alors pour construire des fédérations d'outils faisaient l'hypothèse selon laquelle les outils étaient en mesure de "comprendre" et d'interpréter les concepts de l'univers commun de l'application (par l'invocation de l'UCA et par la réception des évènements provenant de ce dernier). En réalité, les outils n'ont aucune connaissance a priori de l'univers commun pas plus qu'ils ne sont capables d'utiliser les mécanismes d'interaction (serveur de message, technologies permettant d'effectuer des appels de méthodes à distance, etc.).

Concernant le scénario du transfert de produit, les concepts de l'application sont ceux du méta-modèle d'APEL (par exemple) : *activités*, *produits*, *ressources*, etc. Les outils de cette fédération sont : un gestionnaire d'espaces de travail, un outil de traitement de texte, deux gestionnaires de versions. Les concepts qu'ils utilisent sont singulièrement différents les uns des autres (le concept de *document* pour l'outil de traitement de texte, le concept de *fichier* pour les autres outils).

Jusqu'alors, tous les outils que nous décrivions étaient, en fait, des outils abstraits dans le sens où ils satisfaisaient à l'expression de nos besoins (en terme de compréhension des concepts de l'UCA, de technologies permettant la communication, etc.). Ceci nous permettait de définir des fédérations indépendamment de l'implémentation réelle des outils, de leurs caractéristiques intrinsèques, de leur localisation sur le réseau, etc.

Bien sûr, la réalité est toute autre et l'un des problèmes qu'il nous faut résoudre est la correspondance entre les outils abstraits, que nous avons appelés des **rôles** (section IV.3.6), et les outils du monde réel, ceux qui implémentent les services définis par les rôles.

Nous allons, dans la suite de cette section, montrer comment nous parvenons à résoudre ces problèmes de non concordance et d'incohérence.

IV.4.1.2 L'incohérence au niveau des méta-modèles

Reprenons l'exemple du transfert de produit. Nous définissons, pour cette fédération et pour ce scénario particulier, un rôle appelé *GestionnaireDeProduit* impliqué lors d'un transfert de

produit entre activités. Ce rôle fournit un service "`transfert(Product p)`" permettant de transférer un *produit p* d'une activité vers une autre. Dans cette fédération, nous recensons plusieurs outils parmi lesquels RCS, CVS, un outil de traitement de texte, etc.

Par rapport à l'application définie, nous choisissons l'outil RCS pour endosser le rôle du *GestionnaireDeProduit*. Il n'y a pas de concordance entre le méta-modèle du rôle *GestionnaireDeProduit*, gérant des *produits* et le méta-modèle de RCS, gérant des *fichiers*. Les concepts de chacun sont différents. Il est important d'avoir des moyens permettant d'effectuer la concordance.

Nous définissons un niveau d'adaptation entre les outils et l'UCA. Ce niveau d'adaptation est chargé de faire les correspondances entres les différents rôles définis dans le cadre d'une fédération particulière et les outils qui réalisent ces rôles.

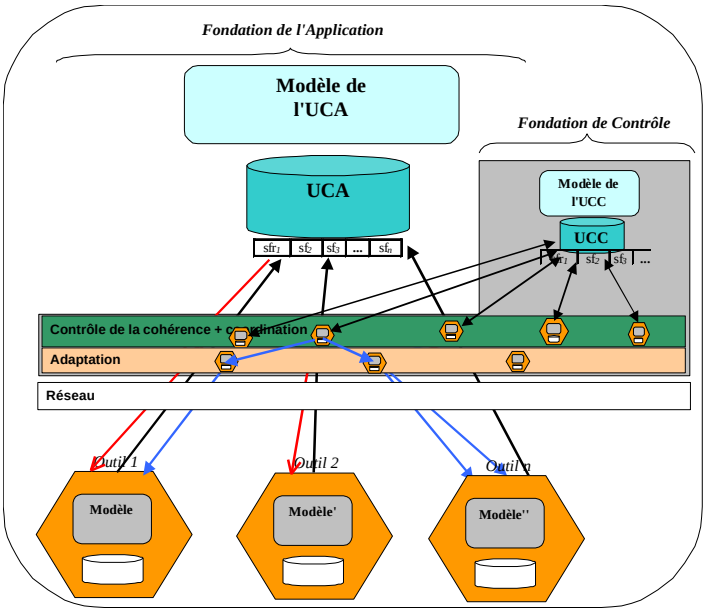

Figure IV.8 Gestion de la concordance

Le niveau d'adaptation contient des composants particuliers de la fondation de contrôle que nous appelons des **représentants**[45]. Un représentant, pour un outil donné, établit les correspondances entre les concepts et services de l'outil et les concepts et services déclarés dans les définitions des rôles endossés par l'outil. Il effectue donc les concordances syntaxiques et sémantiques nécessaires.

Ce niveau d'adaptation fait logiquement partie de la fondation de contrôle puisqu'il utilise des informations de l'univers commun de contrôle.

Dans l'exemple fourni, le représentant de l'outil RCS va faire la correspondance entre le service du rôle *GestionnaireDeProduit* "`transfert(Product p)`" et la commande "`CSRCS checkIn nomFichier`"[46] connue de RCS (où *nomFichier* est le nom du fichier dont il faut faire une révision).

La même question se pose dans le sens inverse, à savoir les conversions entre les concepts de l'outil et ceux de l'UCA. Dans le cas général, ceci ne peut pas être résolu facilement [Estublier et Verjus 2001, Estublier et al. 2001b] ; en effet, si les concepts de l'outil sont de bas niveau alors que ceux de l'UCA sont d'un plus haut niveau, la traduction dans ce sens est bien plus compliquée que la traduction inverse. En reprenant l'exemple, il n'est pas trop compliqué

[45] Le terme anglophone utilisé est "proxy".
[46] Il s'agit de la commande en ligne à invoquer sous un interpréteur de commande (MS-DOS ou l'interpréteur de MS-Windows 2000).

d'effectuer la correspondance entre un *produit* et un *fichier*, contrairement à la traduction inverse[47]. Cette dernière ne peut être réalisée que si nous pouvons définir des conventions entre les deux concepts. La traduction s'effectuera bien souvent manuellement sauf s'il existe des opérations prédéfinies et stockées dans une bibliothèque mise à la disposition des représentants.

Le représentant remplit les fonctions principales suivantes :

- effectuer les traductions entre l'UCA et l'outil et l'inverse lorsque cela est rendu possible,
- mettre en place et établir la communication avec l'outil dont il est le représentant en utilisant les mécanismes d'interaction de la fédération.

Un outil dispose d'un seul représentant pour une fédération donnée même s'il est amené à endosser plusieurs rôles dans une même fédération. L'intérêt des représentants est de préserver une autonomie forte des outils.

Pour ces raisons, à la fois de complexité et de conversions, les représentants ne peuvent pas être des composants générés automatiquement en partant de la description de la fédération. La définition des rôles d'une part, et celles des outils d'autre part, permet tout de même de faciliter la tâche du concepteur (voir le cinquième chapitre).

IV.4.2 L'adaptation technique

Parallèlement à l'intégration conceptuelle, les outils doivent pouvoir être techniquement intégrés à la fédération. Cela veut dire que la communication entre les outils et l'univers commun de l'application soit garantie, permettant les sollicitations bi-directionnelles et ce, dans un contexte distribué. Pour satisfaire ces besoins, nous allons associer, à chaque participant, une **façade**[48] dont le rôle est l'intégration technique de son (outil) participant.

En reprenant les problèmes d'incohérence des concepts, de synchronisation des états (local de l'outil avec l'ACU et dans les deux sens), de cohérence, etc., nous identifions deux points qui restent à traiter :

- l'invocation de l'outil depuis son représentant,
- le synchronisation de l'état de l'ACU en fonction de l'état local de l'outil (nous appelons l'ensemble du processus *l'observation* de l'outil).

Nous allons, pour ce faire, adjoindre à l'outil une façade dont le rôle est de prendre en compte les deux fonctionnalités sus-mentionnées.

IV.4.2.1 L'invocation de l'outil

Nous avons mis en évidence un problème lié à l'invocation de l'outil depuis son représentant. Ce problème résulte des contraintes liées à la communication via le réseau entre le représentant et l'outil et à l'impossibilité d'invoquer directement les services de l'outils (notamment dans le cas de COTS).

Dans le scénario du transfert de produit et en reprenant le cas de l'outil RCS, seules des commandes en ligne permettent l'invocation de l'outil. Il n'est pas possible, à partir d'une machine distante d'invoquer directement ces commandes : il faut passer par un format d'accessibilité intermédiaire. La façade doit rendre accessible au "monde extérieur" les

[47] Il est en effet peu évident, à partir de la description d'un fichier, d'obtenir une description d'un concept de *produit*.
[48] Le terme anglophone utilisé est "wrapper".

services de l'outil. Pour cela, elle contient du code qui enveloppe les services de l'outil dans leur format d'origine (commandes en ligne, messages, opérations, etc.) et les présente sous une forme permettant leurs invocations à distante (par exemple, description en IDL, interface distante Java/RMI, etc.).

Dans le cas des scénarios, nous avons utilisé l'invocation d'objets distants avec Java/RMI. Chacun des services au format de l'outil est alors encapsulé et décrit à l'intérieur d'une des interfaces distantes (concept d'interface du langage Java). La façade est alors un composant construit à partir d'objets répartis/distants. L'obtention de la façade est fonction des mécanismes et technologies d'interaction. Ainsi, la commande en ligne "`CSRCS checkIn nomFicher`" est encapsulée dans une méthode Java qui accède à l'interpréteur du système d'exploitation permettant l'invocation des commandes en ligne (il en va de même pour les scripts unix, windows, etc.). Cette méthode Java est déclarée dans une interface distante permettant l'accès de ce service à distance (après compilation et génération des amorces[49] clientes et serveur).

IV.4.2.2 L'observation de l'outil

Il s'agit en fait, pour la façade, "d'observer" le comportement de l'outil ainsi que son environnement et de voir, en fonction de ces derniers, s'il y a lieu d'effectuer une synchronisation de l'UCA. Par exemple, une portion de l'itinéraire est annulée (localement) par un outil/système de la fédération : l'ACU doit être informé que l'objet *ticket* doit être supprimé.

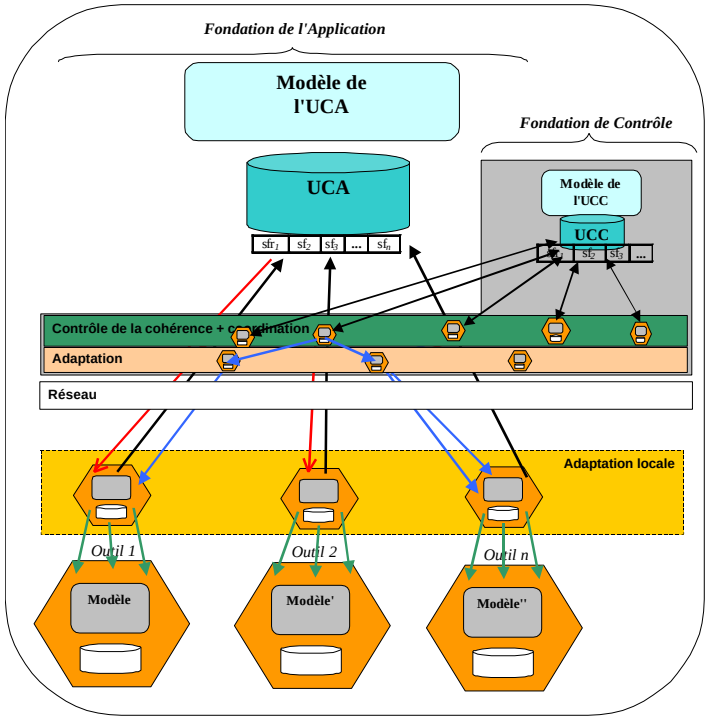

Figure IV.9 Les façades des outils

Il faut alors, pour l'observation, définir les entités gérées par l'outil qui vont devoir être prises en compte, observées, par sa façade.

Dans le cas général, les façades ne peuvent être entièrement et automatiquement générées. L'intervention humaine sera nécessaire pour définir les détails d'implémentation liés aux

[49] Nous trouvons dans la littérature les termes *stub* pour l'amorce du côté client et *skeleton* pour l'amorce du côté du serveur.

spécificités de l'outil ; nous verrons, dans le chapitre suivant, comment notre démarche permet de faciliter et d'aider à leurs obtentions.

IV.4.3 Couplage du représentant et de la façade

Le couple représentant-façade constitue un des mécanismes importants pour la mise en oeuvre des fédérations et l'ensemble des fonctionnalités requises et tout particulièrement en ce qui concerne l'observation de l'outil par la façade.

En effet, l'observation telle que nous l'avons définie repose sur le fait que la façade est à même d'observer l'outil et son environnement et, sur l'état de certains "signaux" se produisant soit au niveau de l'outil, soit au niveau de son environnement immédiat, déclencher une synchronisation de l'état de l'UCA.

Nous proposons d'adjoindre au représentant, un système de persistance dans lequel seront stockés tous les objets de l'outil et de son environnement qui devront être surveillés[50] dans le cadre d'une fédération. Cette définition peut être faite par le concepteur de la fédérateur mais doit pouvoir être construite dynamiquement.

Dans le cas du scénario de transfert de produit, le représentant va gérer une correspondance entre l'entité de l'ACU (le produit) et l'entité de l'univers de l'outil (le fichier). Lors de la première synchronisation, cette correspondance sera initialisée et le représentant "demandera" à la façade de l'outil de "surveiller" le fichier en question. Ainsi, sur chaque modification de l'état local du fichier, la façade "sait" qu'elle doit en informer le représentant qui aura la charge de synchroniser l'ACU.

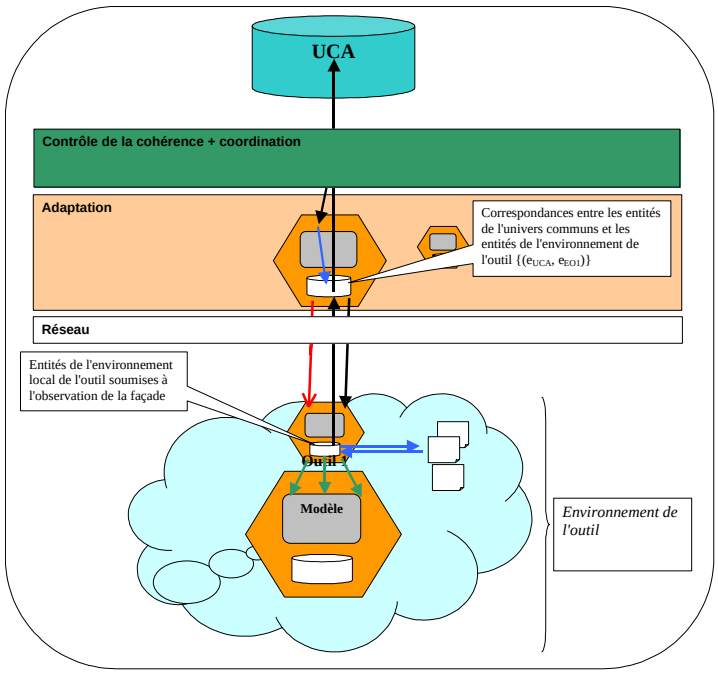

Figure IV.10 La communication représentant-façade

Il y a donc une coopération entre le représentant et la façade pour mener à bien les synchronisations.

La façade doit être en mesure de détecter les changements d'état des entités qu'elle doit surveiller. Dans le cas du document de spécifications, la façade doit surveiller le système de

[50] C'est-à-dire, l'ensemble des objets pertinents et influents sur la fédération qu'il faut surveiller.

fichier sur lequel est stocké le document[51] et peut ainsi savoir lorsque le fichier va être en cours de lecture, etc. (ce système de surveillance est basé sur l'utilisation des services du noyau du système d'exploitation).

Nous devons distinguer :

- les (demandes de) changements d'état de l'UCA émanant des outils,
- les changements des états locaux des outils pour une re-synchronisation de l'UCA.

Il est important de voir que l'observation peut être également vue comme une composante du contrôle. L'outil, du fait de son autonomie locale, peut décider unilatéralement d'opérer des changements localement qui peuvent avoir (ou non) un impact sur l'ACU (l'état global de la fédération). La fédération ne peut pas intervenir localement et donc, va devoir, dans certains cas, refléter le changement au niveau de l'ACU sans effectuer de contrôle sur les contraintes.

En effet, dans l'exemple du scénario de gestion du ticket de transport, un outil peut décider unilatéralement de supprimer une portion de l'itinéraire. Cet acte "volontaire" et local est simplement signalé à l'ACU ; dans ce cas, la façade avertit l'ACU. En revanche, une variante du même scénario est la "demande", émanant de l'outil de changer l'état du ticket de transport (entité de l'UCA)... le changement d'état (effectif) ne se produisant qu'après vérification de l'ensemble des contraintes et si aucune interdiction n'a été constatée (niveau de contrôle). Si aucune contrainte n'a été décelée, la demande est accordée.

Dans le cadre de cette thèse, l'autonomie locale des outils est préservée : nos fédérations se comportent comme des fédérations d'Etats (ces derniers respectent des contraintes de fonctionnement dans le cadre de la fédération mais, sont tout de même souverains de la politique qu'ils appliquent localement). Dans le cas général, nous réfléchissons actuellement sur les moyens à mettre en oeuvre pour contrôler localement les outils depuis la fondation de contrôle. En effet, en reprenant l'exemple ci-dessus, la suppression d'une portion de l'itinéraire décidée localement devrait pouvoir être contrôlée si nous souhaitons garantir la cohérence globale de l'application (de la fédération) et éviter les actions locales non souhaitées.

Dans quels cas met-on en place le mécanisme d'observation ?

Imaginons qu'après la révision de *f1* effectuée par RCS, *f1* soit modifié localement (par un outil – éditeur de texte - de l'environnement de *f1* qui n'appartient pas à la fédération). Dans ce cas, l'entité *p1* de l'UCA n'est plus synchronisée avec l'état local de l'outil. Cette modification locale pourrait entraîner une synchronisation de l'UCA. Nous pensons que dans le cas général, la modification locale n'a pas à être portée à la connaissance de la fédération. En effet, cette modification n'a d'implication que localement : seule une révision de *p1* (donc de *f1*) a été demandée à RCS. Si un outil tiers (de l'environnement de O1) vient à modifier l'état de *f1*, cette modification ne doit (dans le cas général) avoir aucun impact sur l'UCA (sinon, cela voudrait dire que l'outil tiers aurait dû être invoqué dans l'aspect en charge du transfert et donc, faire partie logiquement de la fédération). La modification n'a d'impact que localement.

Dans l'exemple du scénario de la gestion du ticket de transport, le fait qu'un outil supprime localement une portion de l'itinéraire devrait entraîner la suppression du ticket "commun" (ce choix appartient au concepteur de la fédération) et donc, par enchaînement des actions, l'envoie des notifications vers les outils de la fédération. Dans notre cas, il n'est nullement besoin de vérifier si des interdictions ont été définies et si elles doivent s'appliquer car l'acte de suppression locale a déjà été effectué (garantie de l'autonomie locale de chaque outil) ; la modification de l'état d'une entité locale n'a pas forcément à être notifiée à l'univers commun

[51] A partir de la première demande de révision, la façade "sait" à quel endroit du système de fichier se trouve le fichier souhaité (le chemin du fichier fait partie des informations gérées par le représentant).

de l'application. Cela dépend de l'impact et de l'implication de l'entité locale e_{EO1} dans la gestion de l'entité commune e_{UCA} (en fait, la "part", la dimension de l'entité locale dans l'entité commune[52]). Cela dépend de la "confiance" que l'on met dans le comportement individuel des outils et de leurs environnements. Cependant, dans le cas général, nous pensons que les outils doivent être totalement contrôlés ce qui remet en cause l'hypothèse que nous avons faite sur l'autonomie des outils (voir sixième chapitre). Dans le cas du ticket de transport (contrairement au cas du transfert de produit), il est bien clair que l'état local a une importance sur l'état commun dans le sens où une annulation d'une portion d'un itinéraire doit entraîner la synchronisation de l'état commun (l'état de l'UCA par suppression du ticket "commun")... Sans cette synchronisation, l'entité commune *ticket* n'a plus de sens. Or, soit cette synchronisation est effectuée à l'initiative de l'outil qui vient de supprimer localement la portion en question, soit il faut mettre en place un moyen permettant de capturer l'acte de suppression locale pour une synchronisation de l'UCA.

Dans le cas général d'outils de type COTS, et puisque par hypothèse ces outils ne font appel à aucun service externe (cas de l'autosuffisance), le mécanisme d'observation décrit précédemment va permettre de détecter la suppression locale et d'effectuer une post synchronisation de l'UCA lorsque cela est nécessaire. La mise en place du mécanisme d'observation peut se faire selon deux visions :

- une "vision (purement) technique" où il s'agit d'un moyen de fournir aux outils des capacités techniques qu'ils n'ont pas ;
- une "vision contrôle" où ces moyens permettent d'espionner les outils et de les contraindre à faire les synchronisations qu'ils seraient "censés" faire de leur propre initiative.

Le choix de l'observation pour une post synchronisation ne peut être définie de manière automatique et incombe au concepteur de la fédération.

Le concepteur de la fédération devra identifier ce qui est couvert par la fédération de ce qui ne l'est pas (ce qui reste du domaine de l'autonomie des outils), et définir quand les synchronisations doivent être effectuées.

IV.4.4 La correspondance entre les rôles et les outils

L'ensemble des fédérations que nous prenons en compte sont modélisées en deux étapes :

- la première étape consiste à définir une fédération selon un point de vue abstrait, à savoir la définition des concepts de l'application, les buts de la fédération, les rôles, les devoirs et les interdictions des différents rôles, les paradigmes de contrôle, etc. Il s'agit donc de définir le "modèle comportemental" des outils dans le cadre d'une fédération ;
- la seconde étape a pour but d'associer les outils "réels" aux rôles qui ont été définis.

Une fédération donnée sera conforme à sa description abstraite à la condition que les outils participants se comporteront comme les rôles décrits dans le modèle de la fédération.

Dans le cas général, le représentant d'un outil implémente les interfaces décrites par chacun des rôles que doit endosser l'outil qu'il représente ; il invoque l'ACU (au travers de son API) et est invoqué par l'ensemble des aspects de l'UCA pour lesquels il doit contribuer (via la description des rôles et si le modèle de la fédération contient des aspects).

[52] Souvenons-nous que l'entité commune n'est qu'une entité abstraite dont la réalité n'est matérialisée que par l'ensemble des entités locales associées à l'entité commune.

Cependant, il est possible que dans certains cas, l'utilisation du représentant ne soit pas requise ou ne soit pas souhaitée. C'est le cas d'une fédération dont le mode de fonctionnement suit le paradigme de l' "anarchie" : les outils peuvent invoquer directement l'univers commun et recevoir les notifications ; ou alors, le cas où l'outil ou/et sa façade peuvent supporter les prérogatives du représentant (ce qui induit de définir une dépendance entre la façade ou l'outil et une fédération spécifique).

IV.5 Architecture générale pour construire des fédérations d'outils

IV.5.1 Composition d'une fédération

Plusieurs fondations ont été présentées dans les sections précédentes permettant de construire et de contrôler des fédérations d'outils. Nous définissons donc une fédération F comme :

$$F = \{FA, FC, FP\}$$

avec :

- FA la fondation de l'application (elle existe toujours) ;
- FC la fondation de contrôle (elle peut ne pas exister) ;
- FP la fondation du processus (elle peut ne pas exister).

Suivant la définition d'une fondation donnée dans la section IV.2.3, les outils de l'application (appelés aussi outils de la fédération) sont ceux de la fondation de l'application. Les outils de la fondation de contrôle permettent (au moins) d'interpréter les modèles de la fédération permettant de mettre en oeuvre la "politique" et le contrôle de la cohérence de la fédération. Les outils de la fondation du processus permettent (au moins) d'interpréter le processus.

IV.5.2 La fondation de l'application

Elle définit l'application a réaliser et les outils participant à la réalisation de l'application. Elle ne définit aucun mode de fonctionnement, aucune règle ni contrainte pouvant influer sur le comportement de chacun des outils.

IV.5.3 La fondation de contrôle

Au contraire de la précédente, cette fondation a pour but de contrôler le fonctionnement de la fédération en définissant et en contrôlant le comportement de chacun des outils.

Pour cette fondation, nous avons identifié plusieurs niveaux (voir Figure IV.11) :

1. un univers commun de contrôle contenant l'ensemble des modèles de contrôle ainsi que le modèle des outils de la fédération (appelé modèle de composant) ;
2. un niveau de contrôle des initiatives permettant de définir des interdictions pour chaque outil de la fédération ;
3. un niveau de contrôle de la cohérence et de coordination qui définit la politique de coordination des outils en vue de réaliser les objectifs de la fédération ;

4. un niveau d'adaptation qui est lui-même divisé en deux :
    a. le niveau des représentants qui effectuent les conversions nécessaires entre les univers de discours locaux des outils et l'univers commun de l'application ;
    b. le niveau des façades permettant l'invocation des outils à distance et l'observation de ces derniers (ce niveau ne faisant pas réellement partie de la fondation de contrôle).

Reprenant la définition générale d'une fondation, nous identifions :

- l'univers commun (aux outils de cette fondation),
- l'ensemble des outils de cette fondation, répartis au sein des différents niveaux ;
- des mécanismes d'interaction entre les outils de cette fondation et son univers commun.

La fondation de contrôle est, selon la définition de la fédération donnée par [Robert 1989], l'entité commune, l'autorité sous laquelle sont placés l'ensemble des participants.

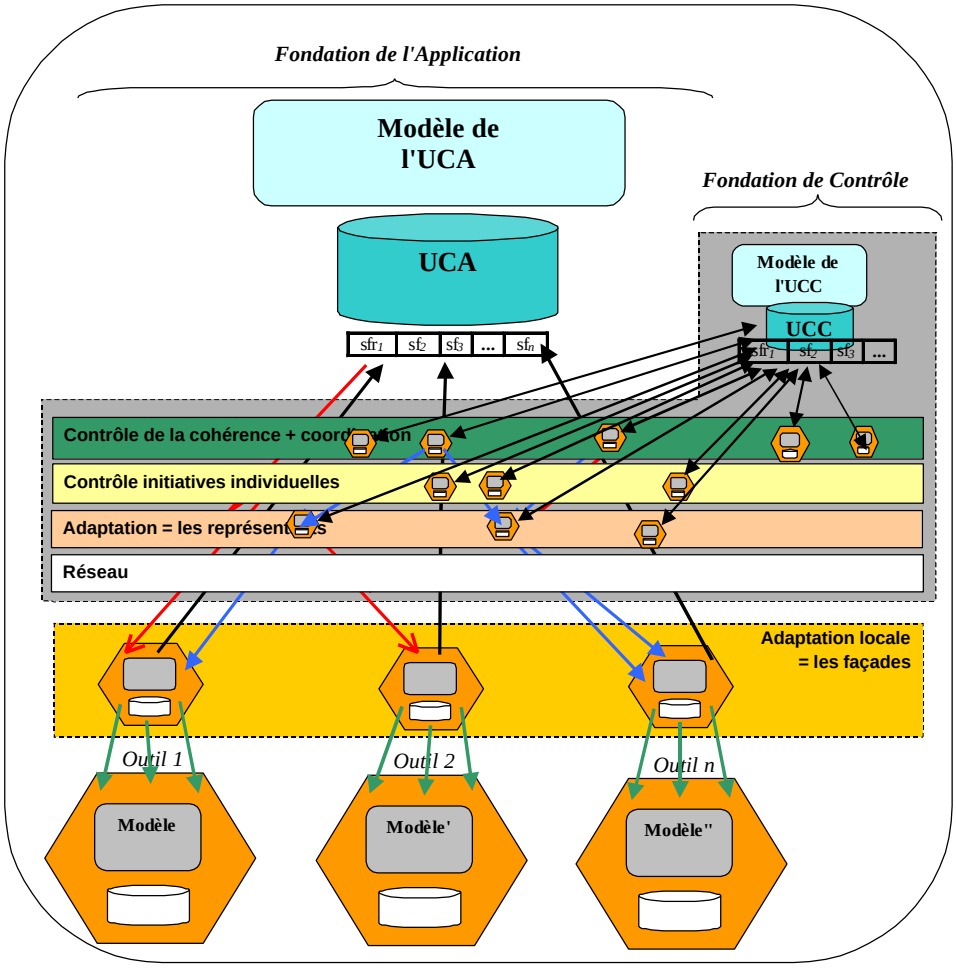

Figure IV.11 Architecture complète de la fondation de contrôle

Le niveau adaptation a une position un peu particulière dans l'architecture que nous proposons, notamment en raison de ses fonctionnalités. En effet, les représentants font partie de la fondation de contrôle de la fédération contrairement aux façades. Les façades auraient pu faire partie également de la fondation de contrôle ; chaque façade pourrait, par exemple, intégrer les prérogatives de contrôle de l'outil auquel elle est associée. Dans ce cas, la façade pourrait inhiber certaines invocations de l'outil, respecter la description des rôles endossés par

l'outil, etc. En quelque sorte, le contrôle de la fédération aurait pu être délocalisé au niveau des façades. Dans l'approche que nous avons choisie, les façades sont au contraire déchargées du contrôle. Les arguments qui plaident pour ce choix sont les suivants :

- le découplage entre les prérogatives de contrôle et les prérogatives liées à l'adaptation de chaque outil (permettant son invocation à distance : implémentation du protocole de communication, utilisation du bus logiciel, etc.) ;
- l'autonomie des outils selon deux facettes :
    1. la façade d'un outil étant géographiquement proche de l'outil, le concepteur de la fédération ne peut la contrôler totalement. Rien n'empêcherait une tierce personne de modifier une façade et de détourner le fonctionnement de l'outil dans la fédération des rôles qu'il doit jouer et des contraintes pesant sur lui, dans le cas où la façade intégrerait du contrôle ;
    2. la préservation de l'autonomie de l'outil qui nous fait agir bien en amont de ce dernier.

Pour cette raison, les façades n'ont pas de lien avec l'univers commun de contrôle et n'intègrent aucune prérogative de contrôle.

Il est important de noter que la fondation de contrôle prend en charge les propriétés requises par l'application comme :

- le respect de contraintes temporelles ;
- la gestion des transactions.

Ces deux propriétés sont définies au niveau "coordination" et interviennent dans la définition des *aspects* liés aux *services fédéraux*. La gestion des transactions ne peut se concevoir indépendamment des propriétés de chaque outil. Dans le cas général, le problème des transactions distribuées n'est pas totalement résolu. Notre proposition n'a pas pour objectif principal d'apporter une réponse au problème mais de prendre en considération les transactions distribuées par la mise en place d'un mécanisme adapté. Ainsi, si l'ensemble des services fournis par des outils et impliqués dans un *aspect* sont de type "transactionnel", alors, l'aspect pourra être défini comme un *aspect* transactionnel[53]. Un service est de type transactionnel si l'outil qui le fournit dispose des moyens garantissant les propriétés ACID[54] et implémente le mécanisme de validation à deux phases (two-phase-commit) et entre autre, s'il peut effectuer un retour-arrière si les conditions l'exigent (annulation de l'aspect en cours d'exécution). Dans ce cas, le concepteur s'appuie sur les caractéristiques propres aux outils qui interviennent dans un aspect[55].

IV.5.4 La fondation du processus

La fondation du processus permet de contrôler l'évolution de l'univers commun de l'application. Dans le cadre de cette thèse nous utilisons cette fondation comme un composant de la gestion du contrôle. Les technologies et les approches permettant son obtention sont présentées dans le second chapitre. Nous ne détaillerons pas davantage son fonctionnement dans la suite de ce document.

[53] La prise en compte des transactions par la fondation de contrôle suppose (1) que les différents services sollicités dans un aspect transactionnel soient transactionnels et (2) que la fondation de contrôle prenne en charge à son niveau la gestion des transactions, de bout en bout.
[54] ACID (Atomicité, Cohérence, Isolation, Durabilité)
[55] La description d'un outil comprend, pour chacun des services fournis par l'outil, son mode d'invocation supporté (transactionnel ou non).

IV.5.5 Architecture générale à trois fondations

En résumant ce que nous avons présenté jusqu'ici, nous avons recensé trois fondations qui "coopèrent" en vue de former et de contrôler une fédération. Nous dégageons plusieurs remarques :

- les outils de type COTS (et bon nombre d'autres d'ailleurs) doivent souvent être associés avec une façade pour interopérer avec d'autres outils ; il en va de même pour qu'un outil, selon une approche boîte noire, puisse participer à une fédération ;
- la façade permet à l'outil avec lequel elle est associée d'utiliser les mécanismes d'interaction de la fondation (de la fédération) de l'application ;
- l'architecture présentée (voir Figure IV.12) montre un découplage entre les trois fondations et leurs rôles respectifs dans la fédération. Cette architecture flexible permet donc d'implémenter différents modes de fonctionnement de la fédération (différents paradigmes) ;
- les mécanismes de contrôle agissent en dehors (en amont) de l'environnement de chacun des outils de l'application et sont donc complètement transparents (préservant ainsi complètement leur autonomie).

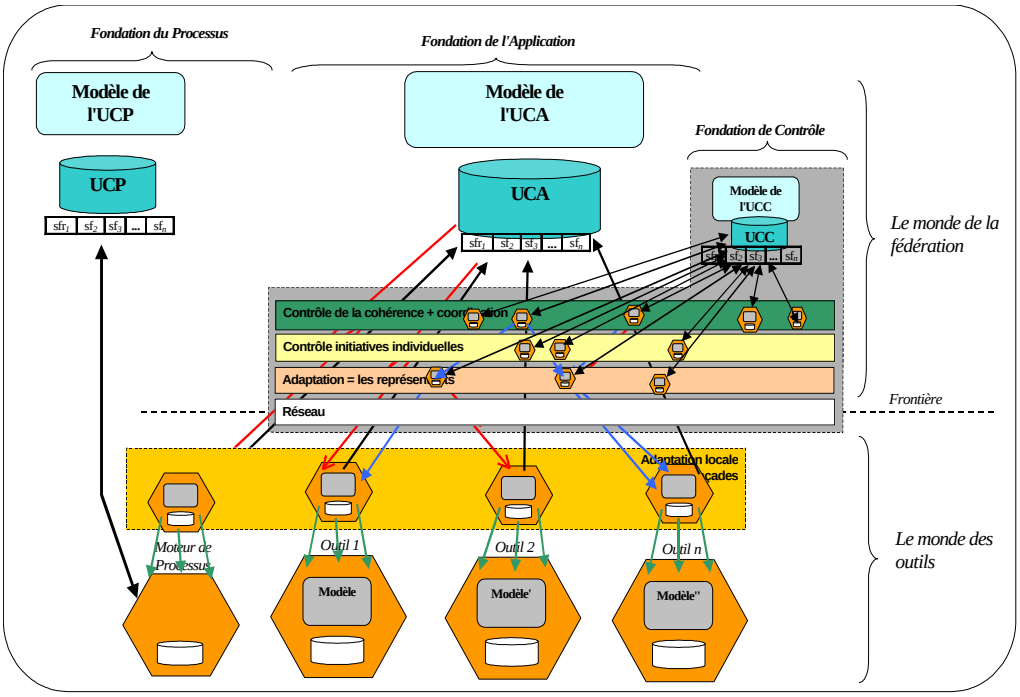

Figure IV.12 Fédération à trois fondations (d'après [Estublier et al. 2001b])

Nous remarquons la position singulière du moteur de processus. Dans la fondation du processus, cet outil est un composant bien connu d'un EGLCP (voir second chapitre). En revanche, il peut ou non être un participant de la fondation de l'application. Dans l'affirmative, cela veut dire que le moteur de processus participe à la l'application (cas de la figure IV.12) ; il est donc soumis au contrôle mis en place comme n'importe lequel des autres outils de la fondation de l'application.

Une fondation telle que nous l'avons définie, n'est finalement qu'une application logicielle ; vue autrement, toute application logicielle est une fondation dans le sens où une application logicielle a un univers commun (qui est souvent implicite), un ensemble d'outils qui seront

appelés, dans le cas des fondations, des composants ou des modules (morceaux de code) et des mécanismes d'interaction permettant aux composants d'interagir.

IV.5.5.1 Transparence de la fondation de contrôle

L'un de nos objectifs est de rendre la gestion (fonctionnement, contrôle, etc.) de la fédération aussi transparente que possible des outils qui la composent. En reprenant la définition de la fédération (voir section IV.5.1), nous remarquons que si nous construisons une fédération sans fondation de contrôle ni fondation de processus, il y a identité entre la fédération et la fondation de l'application (la fédération se réduit à une fondation).

$$F = FA$$

Le fonctionnement d'une telle fédération suit le paradigme de l'anarchie. D'autre part, aucun modèle de fédération n'a été défini. Nous rappelons que le rôle de la fondation de contrôle est (1) d'intégrer le modèle de la fédération et (2) de l'interpréter à l'exécution.

Pour que nous puissions respecter l'autonomie des outils, il faut que la fondation de contrôle agisse de façon transparente. Or, d'après la définition de la section IV.2.3 :

$$FA = \{UC_{FA}, O_1, O_2, ..., O_n, MI_{FA}\}$$

où :

- $O_1, O_2, ..., O_n$ sont les outils de la fondation
- MI_{FA} sont les mécanismes d'interaction entre les outils de la fondation et l'UCA

L'une de nos hypothèse est de ne pas affecter les outils. L'idée retenue est de placer physiquement la fondation de contrôle au niveau des mécanismes d'interaction. Nous verrons dans le cinquième chapitre comment cela a été réalisé.

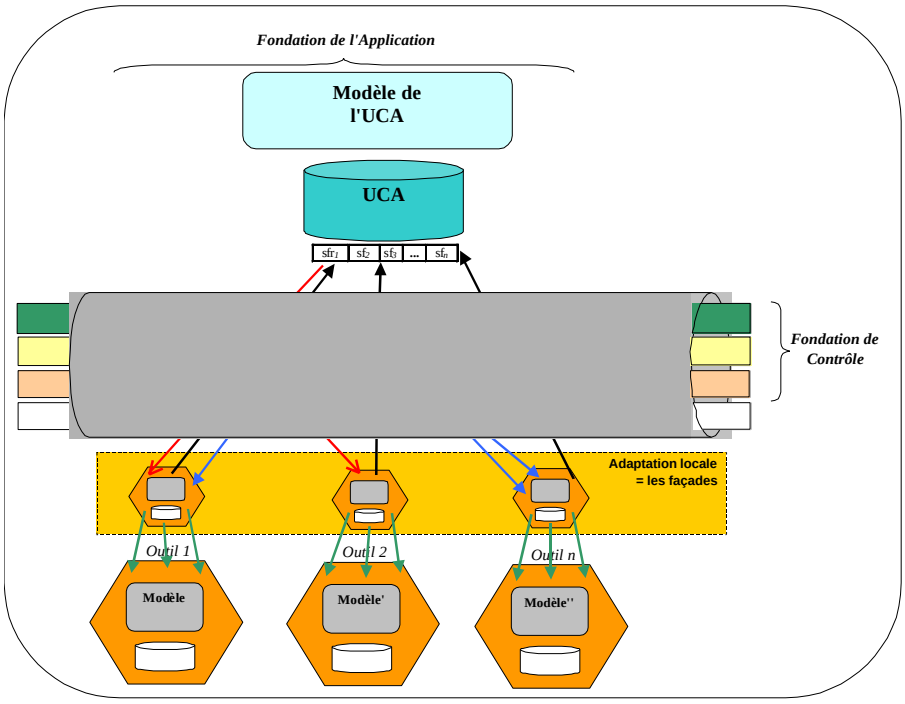

Figure IV.13 Vue logique des différents éléments d'une fédération

Les fédérations d'outils de type COTS font appel aux bus logiciels (middleware), permettant l'invocation de services distants synchrones et/ou asynchrones (voir CORBA, Java/RMI, JMS, etc.). La fondation de contrôle intervient, comme le montre la figure IV.13, entre l'infrastructure de communication (réseau et protocoles) et l'univers commun de l'application.

La fondation de contrôle prend à sa charge l'ensemble des propriétés importantes et couramment rencontrées dans les systèmes distribués (transactions[56], persistance, etc.).

Ainsi, nous montrons que construire une fédération à base d'outils de type COTS et en utilisant des "standards" peut se faire en toute transparence, à l'insu des outils.

IV.5.5.2 Interopérabilité des fondations

L'architecture des fédérations que nous proposons est relativement simple du point de vue des outils. Seul l'univers commun (tout ou partie) est connu des façades des outils, les mécanismes d'adaptation (conversion, etc.), de contrôle, de vérification de propriétés étant à la charge de la fondation de contrôle.

Nous pouvons donc dire que la fondation de contrôle est orthogonale aux autres fondations de la fédération. En effet, toute fondation utilisant les mécanismes d'interaction de la fondation de l'application peut être gérée par la fondation de contrôle à la condition que les outils soient décrits dans l'univers communs de la fondation de contrôle.

La fédération est conceptuellement à géométrie variable suivant ce que le concepteur de la fédération souhaite voir gérer par la fondation de contrôle. Par exemple, la fondation du processus peut être utilisée de façon totalement indépendante. Dans ce cas, les composants de cette dernière implémentent leurs propres mécanismes d'interaction (cette fondation peut être une application monolithique)... Les services de l'application correspondant à la partie "moteur du processus" seront exhibés et feront l'objet d'une façade permettant à l'application d'intégrer la fédération.

Dans le cas général, selon une vision "gros grains", la fondation du processus peut être vue comme une application logicielle "banale" qu'il est possible d'intégrer à la fédération sous respect des conditions précédemment décrites :

1. l'application dispose d'une interface ;
2. il est possible d'associer une façade à cette application utilisant les mécanismes d'interaction de la fondation de l'application de la fédération.

Le concepteur de la fédération doit définir quel sera le comportement de cet outil, cette application, cette fondation dans la fédération qu'il/elle intègre.

Cette vision et l'architecture ouverte que nous proposons pour les fédérations d'outils permettent donc de faire inter-opérer plusieurs fondations, plusieurs applications, plusieurs fédérations en adoptant des approches à granularité variable selon les besoins mais aussi les possibilités.

IV.6 Processus de modélisation des fédérations d'outils

Comme souvent dans la problématique de modélisation, plusieurs approches peuvent être prises pour modéliser des fédérations d'outils :

i. approche montante ;
ii. approche descendante.

Pour ces deux approches, nous identifions les étapes suivantes :

[56] Sous réserve de la capacité des outils impliqués à mettre en oeuvre les mécanismes de gestion des transactions à leur niveau (c'est-à-dire, intrinsèquement).

- la modélisation de l'UCA et des services fédéraux ;
- la modélisation des outils participant à la fédération ;
- la modélisation du contrôle de la fédération ;
- la modélisation de la topologie de la fédération (l'ensemble des machines physiques qui sont impliquées dans la fédération – celles sur lesquelles s'exécutent les outils de la fédération – et leurs caractéristiques) ;
- l'adaptation des outils au contexte de la fédération.

La modélisation du contrôle comprend elle-même :

- la définition des rôles, c'est-à-dire des comportements attendus des outils dans le cadre de la fédération ;
- la définition des aspects, c'est-à-dire la coordination des rôles afin de satisfaire aux besoins d'un service fédéral ;
- l'attribution des rôles, c'est-à-dire l'association des rôles avec les outils réels de la fédération.

La modélisation des outils comprend :

- la définition des modèles d'outils (modèle intrinsèque) ;
- la définition des instances à partir des modèles d'outils ;
- l'attribution des rôles aux instances (aux outils "réels").

Nous allons aborder chacune de ces deux approches et voir leurs avantages et inconvénients respectifs.

IV.6.1 L'approche montante

L'approche montante permet de définir la fédération en partant, d'une part de l'idée que le concepteur de la fédération se fait/a de l'application et d'autre part des outils qu'il a à disposition et sur lesquels il doit s'appuyer. A partir de ces deux données il va (voir Figure IV.14) :

- définir l'UCA qui tiendra compte d'une part de l'application et d'autre part qui doit être l'abstraction de la "connaissance" partagée des outils ;
- définir les modèles d'outils en fonction des outils disponibles ;
- proposer un modèle de contrôle ;
- définir la couche d'adaptation entre les outils du monde réel et l'abstraction qui en aura été faite dans le modèle de contrôle.

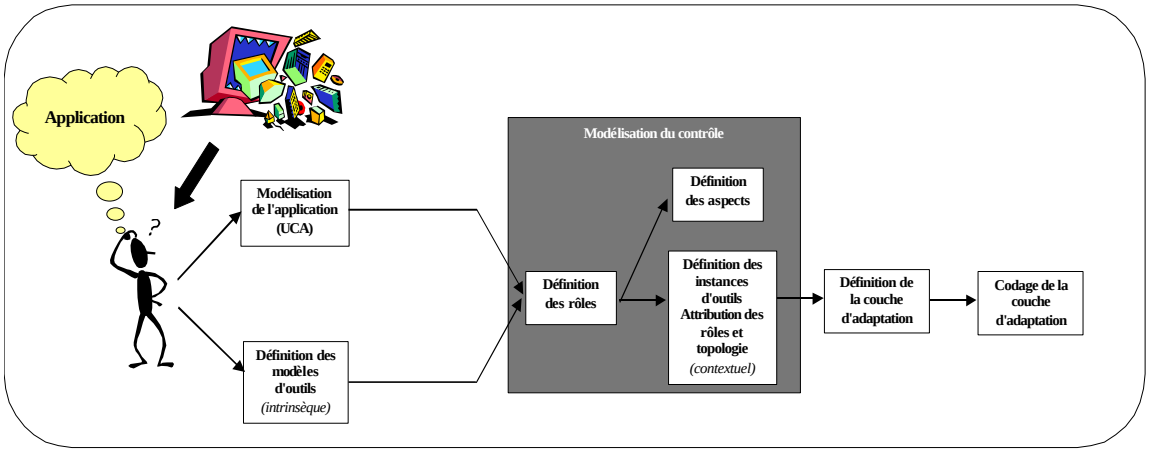

Figure IV.14 Processus de modélisation d'une fédération d'outils selon l'approche montante

Aussi, pouvons-nous qualifier cette *approche de pragmatique* puisque cette approche fait l'hypothèse que les outils disponibles et choisis a priori vont pouvoir satisfaire un grand nombre de besoins et permettent de réaliser une application (proche de l'application "idéale").

IV.6.2 L'approche descendante

Contrairement à l'approche montante, il s'agit pour le concepteur de la fédération en se basant uniquement sur l'application, de proposer un modèle de contrôle, de définir des rôles qui vont lui permettre de sélectionner les outils appropriés. L'objectif de cette approche est de définir un modèle de la fédération sans tenir compte des outils "réels" existants. Le concepteur définit une fédération "idéale" ; cette description permettra d'affiner les besoins de l'application par la proposition d'une vision abstraite des outils : les rôles.

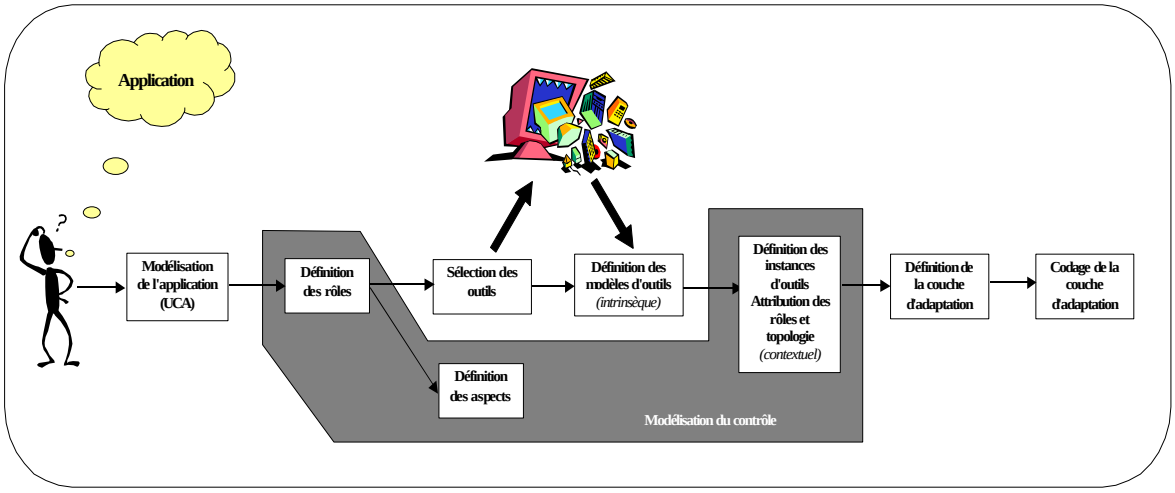

Figure IV.15 Processus de modélisation d'une fédération d'outils selon l'approche descendante

A ce niveau, le concepteur ne se soucie pas des outils réellement disponibles sur le marché. Ceci permet de définir la fédération selon une vision abstraite, sans, dans un premier temps, avoir à ce soucier des détails d'implémentation (les outils qui seront réellement utilisés). En ce sens, notre approche à ce niveau, offre une description abstraite de la fédération qui pourra servir de cahier des charges et dont les informations pourront être exploitées dans la phase d'analyses des besoins, précédant la sélection et l'évaluation des outils en vue de leur sélection [Morisio et al. 2000].

Aussi, pouvons-nous qualifier cette *approche d'idéaliste* (ou *optimiste*), puisqu'elle fait l'hypothèse selon laquelle il existera toujours des outils disponibles sur le marché permettant de couvrir les besoins de l'application.

IV.6.3 L'approche réaliste

Les deux approches citées précédemment diffèrent essentiellement par l'ordonnancement des étapes permettant l'obtention d'un modèle de fédération d'outils.

Dans l'approche *montante*, le concepteur dispose a priori des informations sur les outils participant à la fédération. Cela permet de déduire plus facilement le modèle de fédération, notamment en utilisant les connaissances que le concepteur a des outils pour définir l'UCA. En revanche, cette approche occulte les étapes portant sur la qualification et la sélection des outils de type COTS [Abts 1997, Abts et Boehm 1997, Morisio et al. 2000]. D'autre part, selon cette approche, les outils sont imposés, ce qui peut être un handicap pour concevoir une fédération (notamment l'ensemble des outils n'est pas satisfaisant, c'est-à-dire est incomplet et/ou inadapté aux besoins de l'application).

Au contraire, l'approche *descendante* permet d'avoir plus de souplesse dans le sens où le choix des outils se fait en fonction des contraintes de l'application (seconde partie du processus de modélisation). Il permet d'avoir des outils adaptés. Par contre, cette approche impose au concepteur une modélisation de la fédération selon une vision abstraite, ne tenant pas compte de la réalité des outils. Entre autre, l'UCA ne tient pas compte de la "vision outils" : l'UCA est défini a priori contrairement à l'approche *montante*.

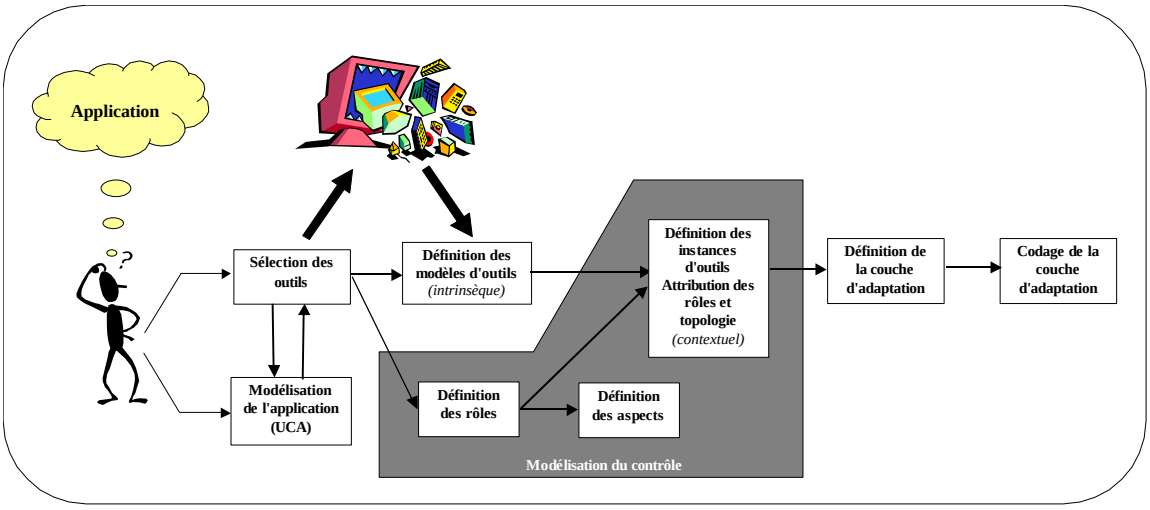

Figure IV.16 Processus de modélisation d'une fédération d'outils selon une approche mixte

Dans la pratique, les expériences montrent qu'il s'agit d'une approche mixte, s'inspirant des deux approches. Le concepteur de la fédération va devoir faire des allers-retours entre le cahier des charges de l'application et les outils disponibles afin de trouver un point d'équilibre (ajustement du cahier des charges en fonction des outils existants et acquisition de nouveaux outils pour couvrir au mieux les besoins de l'application – voir Figure IV.16). Ce point d'équilibre est défini comme suit :

> il s'agit de l'ensemble des outils disponibles sur le marché et/ou des outils déjà disponibles au sein des organisations qui permettent réellement de satisfaire au mieux les besoins de l'application.

IV.6.4 La modélisation des outils : une question de point de vue

L'un des enjeux de cette thèse est de caractériser les participants de la fédération.

En reprenant les spécifications portant sur les composants, leurs définitions ainsi que les modèles proposés, nous définissons, pour un composant [ODP 1995] :

- un modèle de comportement interne (souvent inaccessible dans le cas d'approches de types "boîte grise" et "boîte noire"),
- un modèle de comportement externe (qui fait l'objet des spécifications et des modèles proposés).

Dans le cadre de fédérations d'outils existants (COTS, systèmes et applications patrimoines), une telle dichotomie n'est pas très adaptée notamment du fait qu'il nous est impossible, dans le cas général, de décrire le comportement interne des outils.

Nous pensons que la perception des outils se fait selon deux dimensions :

1. le **modèle intrinsèque** ou **modèle d'outil** qui décrit les outils selon un point de vue externe, en dehors de tout contexte (ce modèle enrichit les spécifications et modèles déjà proposés – EJB, CCM, modèle générique) : il s'agit d'un modèle statique. Il décrit un composant "prêt à l'emploi" ;
2. le **modèle contextuel**, précisant le *comportement* de l'outil dans un environnement particulier et pour une fédération particulière. Il raffine le modèle intrinsèque par l'ajout des contraintes et la description du comportement attendu de l'outil dans le cadre de la fédération.

Dans le cadre des approches décrites précédemment, le modèle d'outil est défini lors de l'étape consacrée à la modélisation des outils alors que le modèle contextuel est obtenu essentiellement lors de l'attribution des rôles.

IV.6.5 Processus pour des fédérations d'outils de type COTS

En reprenant l'approche proposée par [Morisio et al. 2000] et le processus de développement de systèmes à base d'outils de type COTS, nous plaçons les activités du processus (de développement de fédérations d'outils) selon l'approche que nous avons qualifiée de "réaliste" comme le montre la figure IV.16.

Les activités présentées dans [Morisio et al. 2000] ne sont pas toutes mentionnées dans la figure IV.17 par simple soucis de lisibilité et de clarté.

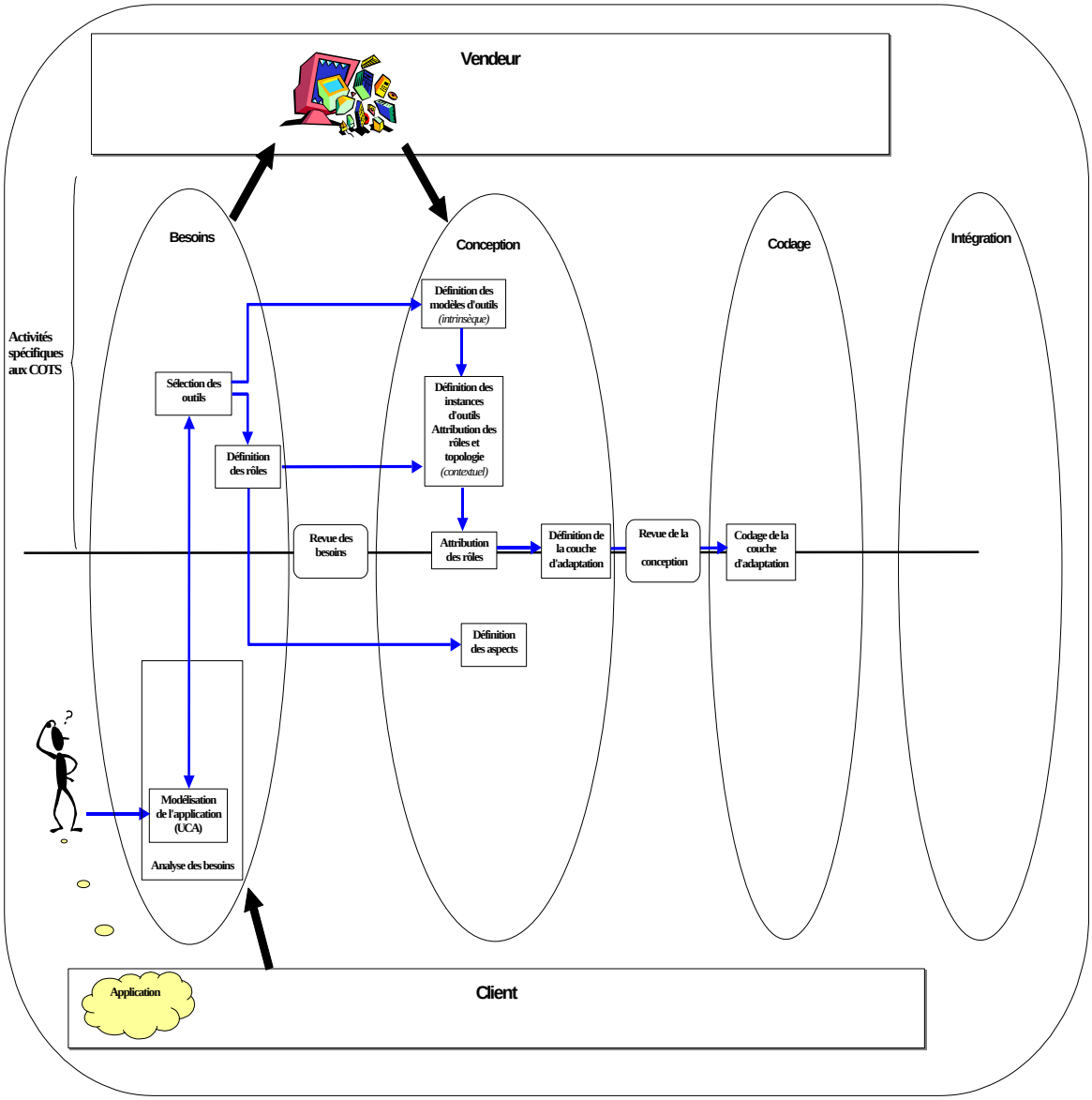

Figure IV.17 Les activités du processus de fédération d'outils

Nous remarquerons que notre proposition s'inscrit majoritairement dans la phase de conception ; cette dernière ayant fait l'objet de peu de travaux [Estublier et al. 2001a]. Comme il est indiqué dans [Morisio et al. 2000] les principaux travaux qui se placent dans la phase de conception sont essentiellement ceux portant sur les architectures logicielles. Or, nous avons montré que, par rapport à nos objectifs, ils présentent un certain nombre de limites (voir troisième chapitre).

Les différents travaux portant sur les processus logiciels [Royce 1970, Boehm 1981, Boehm 1986, Promoter 1984, Promoter 1999] montrent que plus les problèmes sont décelés en amont, plus les spécifications sont précises, plus la conception est complète et rigoureuse et plus les phases avales seront facilitées et dégagées des questions qui auront été traitées dans les phases précédentes. Notre proposition permet, en considérant la construction de fédération d'outils dès la conception, d'alléger les activités des phases de codage et d'intégration.

IV.7 Conclusion

Ce chapitre a eu pour principal objectif l'étude de la construction des fédérations d'outils et de présenter notre approche dans ce contexte. Pour cela, l'idée est de concevoir une fédération comme un ensemble de fondations. Le mode de fonctionnement de la fédération est directement conditionné par :

- la topologie de la fédération, c'est-à-dire la situation de chacune des fondations dans la fédération, avec leurs outils respectifs (nous nous sommes limités aux outils de la fondation de l'application) ;
- le *modèle de fédération* définissant le contrôle de la fédération et basé sur les concepts d'*univers commun*, de *services fédéraux*, de *rôle*, de *modèle d'outils*, d'*aspects*, d'*instance*, etc.

La définition de base d'une fédération s'appuie sur la caractérisation de trois fondations ayant, pour chacune et lorsqu'elles sont présentes, des attributions particulières. Aussi, il est possible, en fonction de la position d'un outil à l'intérieur d'une fédération de savoir à quelle fondation il appartient :

- s'il participe à l'application, il fait partie logiquement de la fondation de l'application ;
- s'il participe au contrôle de la fédération, il fait partie logiquement de la fondation de contrôle ;
- s'il participe au processus de contrôle de l'application, il fait partie logiquement de la fondation du processus.

Cela revient à savoir dans quel univers commun l'outil est impliqué. Ce découplage permet de bien identifier les prérogatives de chacun des outils dans le cadre des fédérations, notamment dans les cas où les fédérations sont complexes.

Dans le cas le plus simple, une fédération peut se résumer à une fondation (de l'application) ; cependant, il n'existera aucun modèle explicite définissant la fédération et les outils y participant puisqu'il n'existe pas, selon ce schéma, d'univers commun de contrôle (intégrant le modèle de fédération).

L'approche que nous avons adoptée rentre dans le cadre des systèmes à base d'outils de types COTS. Nous pensons que nos travaux facilitent les tâches incombant au concepteur de ces systèmes par le découplage des différentes tâches ; ils se concentrent prioritairement sur la phase de conception, facilitant ainsi les phases de codage et d'intégration.

IV.7.1 Une approche selon deux axes

L'approche que nous avons adoptée permet de considérer les fédérations de progiciels selon deux axes :

- l'axe horizontal,
- l'axe vertical.

L'axe horizontal considère le contrôle d'une fédération d'outils selon trois fondations de nature différente (fondation de l'application, fondation du processus, fondation de contrôle) et selon des domaine différents (concepts propres au domaine de l'application, concepts propres au domaine du processus, concepts propres au domaine de la gestion du contrôle). Elles permettent, ensemble, de s'adapter et de satisfaire aux besoins de contrôle. En revanche, ces

fondations reposent sur la même approche (même définition de base, même niveau d'abstraction).

L'axe vertical, au contraire, permet de passer de la vision abstraite d'une fédération (les concepts de l'univers commun de l'application) vers les différentes implémentations de ces concepts (au niveau de chaque outil) en utilisant les concepts permettant de définir une fédération d'outils (concepts d'outils, de rôles, de représentants, de façades, etc.).

Nous pensons que ces deux axes sont une des caractéristiques de l'originalité de nos travaux.

IV.7.2 Topologie et fédération multi-fondations

Nous avons proposé une topologie particulière pour les fédérations caractérisée par la position singulière de la fondation de contrôle : entre les outils de la FA et l'UCA, notamment au niveau des mécanismes d'interaction. Cette topologie est une caractéristique de notre proposition.

Il est tout à fait possible de concevoir une fédération comprenant plusieurs fondations. Dans ce cas nous pouvons définir différentes architectures pour ces fédérations allant du totalement distribué au cas le plus centralisé. Ainsi, une fondation de contrôle peut contrôler plusieurs fondations d'applications et, inversement, une fondation de l'application peut être contrôlée par plusieurs fondations de contrôle, chacune d'entre elles étant dédiée à un fragment particulier de l'UCA.

Imaginons que plusieurs organisations soient impliquées dans la réalisation d'une application, mettant à disposition de la fédération plusieurs outils. Nous pouvons :

- soit considérer une fédération à une FA centralisée,
- soit distribuer l'UCA au niveau de plusieurs sous-fédérations.

Le contrôle de la fédération globale interviendra à un niveau supérieur où seuls les sous-fédérations seront visibles. Le contrôle à l'intérieur de chacun des sous-fédérations est délégué et géré au niveau de chacune des organisations de la fédération globale. Ce type de configuration permet de prendre en compte les contraintes liées d'une part, à l'autonomie des organisations et à la maîtrise des outils qu'elles mettent à disposition (dans la fédération) et d'autre part à la question particulièrement importante de la gestion de la confidentialité (celle-ci, dans ce cas, restant à la charge des organisations impliquées).

IV.7.3 Support à la mise en oeuvre de différents paradigmes

Comme nous l'avons montré, notre proposition permet de construire des fédérations qui permettant à la fois un mode de fonctionnement suivant le paradigme de l'anarchie [Estublier et al. 1998b, Estublier et Verjus 1999, Estublier et al. 2001a] et un mode de fonctionnement suivant le paradigme de la dictature [Estublier et al. 1998b, Estublier et Verjus 1999, Estublier et al. 2001a].

Ces différents modes peuvent être combinés dans une même fédération sachant qu'il est possible de définir un aspect "anarchique" alors qu'un autre pourra être "dictatorial".

Le chapitre suivant va porter sur la mise en oeuvre de fédérations d'outils et, entre autre, les expériences et le prototype développé permettant d'exécuter des fédérations d'outils.

Chapitre V IMPLEMENTATION, PROTOTYPAGE ET EXPERIMENTATIONS

Implémentation, Prototypage et Expérimentations

Les objectifs du quatrième chapitre étaient de proposer un cadre de définition pour la conception et la construction de fédérations d'outils, particulièrement les outils de type COTS. Ce cinquième chapitre a pour but de présenter de façon plus détaillée les concepts nous permettant de modéliser les fédérations d'outils et surtout, de voir comment ils permettent à l'utilisateur de définir des fédérations d'outils.

Ce chapitre abordera également la phase d'expérimentations, les résultats expérimentaux obtenus ainsi que les techniques d'implémentation du prototype qui nous ont permis de mettre en oeuvre des fédérations d'outils.

V.1 Rappels des enjeux

L'un des objectifs majeurs de nos travaux est de concevoir une fédération d'outils, c'est-à-dire, de donner au concepteur (utilisateur) des moyens permettant d'intégrer des outils hétérogènes distribués et autonomes en vue de réaliser, ensemble, une application.

Nous avons vu qu'au sein d'une fédération, les outils sont soumis aux règles d'une autorité commune que nous avons appelée *fondation de contrôle* (voir quatrième chapitre). Cette fondation joue le rôle de "colle" entre les outils et l'univers commun de l'application, et à deux niveaux :

- conceptuel : les représentants effectuent les conversions nécessaires entre les concepts de l'UCA et les concepts propres à chaque outil ;
- technique : les façades permettent aux outils de se connecter à la fédération et de rendre disponibles à cette dernière les services de chaque outil.

Dans les domaines tels que l'EAI, les approches CBSE, etc., le concepteur de systèmes à base de composants ou d'applications logicielles n'a, à sa disposition (voir troisième chapitre), qu'un éventail de choix techniques, sans méthodologie et sans véritable aide à la mise en oeuvre. Qu'apportons-nous au concepteur dans ce contexte pour lui faciliter la tâche, c'est-à-dire faciliter l'intégration des outils ? Comment construit-on des fédérations d'outils ? Ce chapitre présente l'utilisation des concepts introduits dans le chapitre précédent pour définir des fédérations d'outils ainsi que l'implémentation de ces concepts formant la fondation de contrôle.

V.2 Construire des fédérations d'outils

Nous avons mis en évidence, au cours de la présentation de notre approche (quatrième chapitre), plusieurs concepts sur lesquels nous nous basons pour définir des fédérations d'outils :

- les *outils* qui sont les entités logicielles autonomes et participant à la fédération (réalisant l'application) ;
- l'*univers commun* (de l'application), défini par le concepteur de la fédération qui représente le but de la fédération (ce que doivent atteindre ensemble les outils). L'univers commun contient l'abstraction des concepts et des entités de l'application, chacune d'entre elles concernant au moins deux outils de la fédération (entités "communes") ;
- les *aspects* qui sont les éléments intégrant le contrôle de la fédération. Ils implémentent les réactions de la fédération correspondant aux changements de l'état de l'univers commun par sollicitations des outils. L'ordonnancement des sollicitations définit la coordination des outils au travers des rôles ;
- les *rôles*. Les aspects ne permettent pas d'interagir directement avec les outils. D'une part parce que, par hypothèse, les outils peuvent ne pas être directement accessibles et que d'autre part, l'univers commun et les aspects étant des éléments abstraits, il ne serait pas opportun d'introduire un mécanisme d'interaction ad-hoc avec les outils à ce niveau. Les rôles permettent, au contraire, de rester générique dans la définition des fédérations indépendamment des outils "réels" impliqués dans la fédération. Un rôle est un modèle de comportement pouvant être adopté par un ou plusieurs outils de la fédération ;
- les *représentants*, comme leur nom l'indique, sont les représentants des outils pour la fondation de contrôle. D'une part, parce que les outils peuvent être mobiles, temporairement inaccessibles, les représentants permettent justement de répondre à ces situations. D'autre part, les représentants ont la charge de "router" les sollicitations de la couche (aspects notamment) de contrôle vers les outils en effectuant les transformations nécessaires (conversions de paramètres, etc.) entre les concepts de l'UCA et les concepts propres à chacun des outils ;
- les *façades* permettent l'invocation à distance des outils en enveloppant, au besoin, les services de ces derniers. Contrairement aux représentants, les façades n'effectuent aucune conversion ce qui les rend indépendantes d'un univers commun particulier (donc d'une fédération particulière).

Le concepteur de la fédération va devoir définir des fédérations d'outils en utilisant l'ensemble de ces concepts. Leurs implémentations respectives constituent la fédération (voir Figure V.1) et, entre autre, ce que nous avons appelé la *fondation de contrôle*.

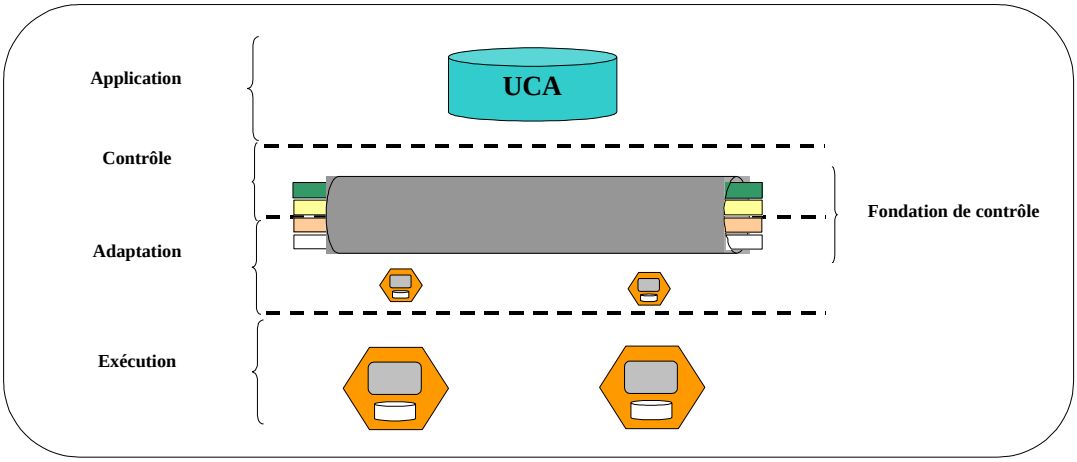

Figure V.1 Les différents niveaux d'une fédération

Selon la figure V.1, notre approche a pour objectif de définir à la fois un univers commun de l'application et une fondation de contrôle nous permettant d'intégrer des outils autonomes : le fruit de cette intégration doit, à l'exécution, atteindre les objectifs de l'application.

Nous allons présenter ces différents concepts et voir comment ils sont utilisés par le concepteur de la fédération, toujours dans l'objectif de mettre en oeuvre des fédérations d'outils.

V.2.1 Définition de l'univers commun de l'application

L'univers commun de l'application est le cœur même de la fédération à partir duquel ont lieu les réactions des outils et sur lequel s'effectuent les actions des outils. Il représente à un instant donné, l'état de (l'abstraction de) l'application.

La définition d'un univers commun dépend entre autre :

- du domaine de l'application ;
- de l'idée que le concepteur se fait de cette dernière ;
- des outils de la fédération (ils influent sur la définition de l'UCA).

La définition de l'UCA doit être relativement exhaustive afin de contenir une représentation de l'application aussi fidèle que possible. Il faut donc pouvoir recenser l'ensemble des concepts pertinents de l'application et pouvant concerner au moins deux outils et pouvoir exprimer ces concepts à l'aide d'un formalisme. La première des tâches demeure une question de recherche et n'est pas chose aisée. Elle reste une activité humaine car elle requiert des facultés intellectuelles qui ne peuvent être automatisées. Cela demande beaucoup d'expertise car il faut avoir une idée complète de l'application (complétude de la connaissance de l'application) et des outils de la fédération pour savoir quels seront les concepts pouvant être communs. Dans la pratique, les concepts retenus sont exprimés par un schéma de base de données. Cette base de données constitue l'univers commun et contient l'ensemble des entités de ce dernier. Comme toute base de données, l'univers commun dispose d'un ensemble de primitives (modification du schéma, ajout/suppression d'objets de la base, etc.).

Nous définissons une interface de l'univers commun contenant un ensemble de services. Ces services appelés *services fédéraux* (voir quatrième chapitre), permettent de manipuler les entités de la base (modifier ou consulter leurs états respectifs – voir Figure V.2). Ces services utilisent les concepts de l'univers communs de l'application (par exemple `deleteActivity(id)`). L'implémentation de ces services va utiliser les primitives de la base

de données (sous forme de requête – `DELETE * FROM activity WHERE activityId="id";` - ou autre).

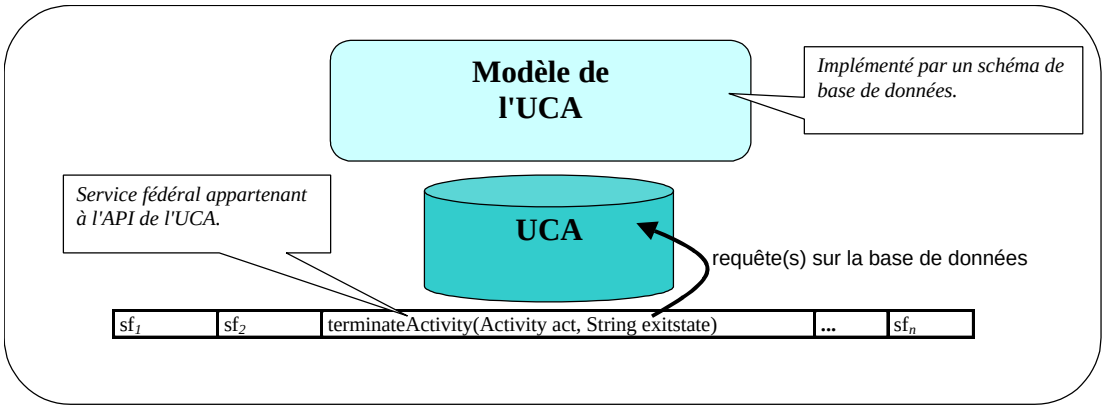

Figure V.2 Un univers commun avec l'ensemble des services fédéraux

Le concepteur, une fois l'UCA défini, va pouvoir l'utiliser au travers de son interface et donc de l'ensemble des services fédéraux. Le contrôle des fédérations repose essentiellement sur le contrôle des invocations de chacun des services fédéraux (quels outils invoquent quels services fédéraux) et le contrôle des souscriptions (ou notifications) en direction des outils (quels outils reçoivent les notifications sur quels changements de l'UCA). Les contrôles sont basés sur la définition d'interdictions (voir quatrième chapitre section IV.3.2).

Cette thèse ne couvre pas la définition de l'UCA pas plus qu'elle n'apporte de moyen ni de méthodologie pour faciliter son obtention. Dans le cas de fédérations "simples", où l'application est clairement définie et où le nombre d'outils (peu complexes) est restreint, la définition d'un univers commun n'est pas très compliquée. En revanche, pour une fédération complexe, intégrant un nombre important d'outils (dont certains peuvent être particulièrement complexes), cette définition peut s'avérer relativement coûteuse.

Nous verrons plus loin dans ce chapitre qu'en ce qui concerne la phase d'expérimentations, l'univers commun n'était pas un SGBDOO (système persistant) mais un ensemble de classes Java (non persistant).

Nous considérons, dans le cas du scénario de transfert de produit, un service fédéral "`terminateActivity(Activity act, String exitstate)`" (voir Figure V.2) dont l'identifiant est "terminateActivity". Ce service utilise de façon implicite la sémantique et les concepts du méta-modèle de processus d'APEL ; sur invocation de ce service, l'activité du processus (stockée dans l'univers commun) dont la référence est un objet de la classe Java `Activity` passe à l'état "terminé" et provoque le transfert de l'objet `doc:spec` de l'activité source vers l'activité cible.

V.2.2 Définition des rôles

Le concepteur de la fédération va définir les rôles nécessaires pour satisfaire les objectifs de la fédération. Présentées dans le chapitre précédent, les phases de définition de l'UCA et celle de la définition des rôles ne sont pas véritablement séquentielles. Au contraire, le concepteur va devoir faire des allers-retours pour adapter la définition de l'UCA aux rôles et vice-versa en fonction des caractéristiques réelles des outils (du marché par exemple). En effet, un rôle est défini dans le but d'être endossé par un ou plusieurs outils réels : définir un rôle sans tenir compte de la réalité (c'est-à-dire des outils réels potentiels pouvant endosser ce rôle) ne présente aucun intérêt. Les différences principales entre la vision du rôle et celle d'un outil réel sont :

- le rôle est exprimé avec les concepts de l'UCA contrairement aux outils ;
- le rôle intègre des prérogatives de contrôle. Ces prérogatives de contrôle sont constituées des *interdictions* portant sur les invocations et les souscriptions des services fédéraux d'une part, et d'autre part des *devoirs* du rôle exprimés en terme de services qui doivent être rendus (proposés). Ces services rendus ne constituent bien souvent qu'un sous-ensemble des services fournis par les outils qui endosseront le rôle en question.

Le premier point permet au concepteur de se reposer sur la définition de l'UCA (et donc de bénéficier du travail effectué lors de cette phase). En effet, les rôles sont définis à partir des concepts de l'UCA que ce soit pour les interdictions ou pour les devoirs.

```
public interface WMRole_I {

        //public void copy(String file, String host);
        public void transfert(Product p);
        public void transfert(String f);
}
```

Interface contenant les services fournis par un rôle (ici *WorkspaceManager*)

Dans le cadre de nos expérimentations, le concepteur de la fédération définit, pour chacun des rôles, une interface (en Java) contenant l'ensemble des services que devront fournir tous les outils endossant le rôle en question. Dans le cas du scénario de transfert, nous avons défini, par exemple, une interface Java `WMRole_I` (voir code ci-dessus) contenant un service "`public void transfert(Product p)`" dont la sémantique est de "transférer le produit – classe `Product` - référencé par *p*". Cela va consister, pour les outils RCS et CVS, à ce que le "produit" soit converti en "fichiers" d'une part (ce qui incombe aux représentants de CVS et de RCS), de transférer physiquement le fichier sur le lieu de chaque outil et, pour ces derniers, d'en faire une révision.

D'autre part, nous verrons dans la suite du document que le concepteur peut, pour un outil donné, lui associer un représentant ou non. Dans le cas où un représentant est associé, ce dernier devra implémenter l'interface du rôle, dans le cas contraire, ce sera la façade de l'outil qui devra implémenter cette même interface.

Un rôle est défini complètement dans une classe Java. Cette classe doit implémenter l'interface décrivant le rôle (classe `WMRole_I`), contenir la définition des interdictions et contenir, pour chaque service fourni, l'invocation de l'outil pertinent. Dans le cas du scénario de transfert (voir quatrième chapitre), si le produit est un document de spécification, l'outil pertinent invoqué est CVS alors que dans les autres cas (autre document), ce sera à RCS d'effectuer la révision.

L'ensemble des interdictions est défini statiquement dans l'implémentation du prototype. Chaque interdiction est une instance de la classe `Prohibition`. Toute modification de la définition des interdictions (donc des rôles) doit donner lieu à une re-compilation des fichiers sources Java.

L'ensemble des services fournis par un rôle est obtenu par introspection de l'interface décrivant le rôle en question.

Une fois les rôles définis, le concepteur de la fédération va devoir définir les participants de la fédération : les outils.

V.2.3 Sélection et définition des outils

L'UCA et les rôles contribuent à la production d'un cahier des charges qui servira de support à la sélection des outils les plus appropriés à remplir les rôles qui auront été définis et donc, à satisfaire les besoins de la fédération. Le cœur même de l'étape de sélection des outils reprend les travaux proposés dans [Kontio 1996, Abts 1997, Abts et Boehm 1997, Morisio et al. 2000].

Dans le cas idéal (approche "descendante", voir section IV.7.2), la sélection des outils devrait se faire une fois que la définition de l'UCA et des services fédéraux d'une part et que la définition des rôles d'autre part aient été achevées. La réalité est tout autre [Morisio 2000] puisqu'en fait, le concepteur va ajuster les besoins de l'application – donc la définition de l'UCA et des rôles – en fonction des outils réellement existants (approche "réaliste", voir section IV.7.3). D'ailleurs, la définition même des rôles tient compte fortement des outils existants.

Ainsi, nous pouvons dire qu'une fois que le concepteur a pu définir un UCA et des rôles et a pu déterminer les outils adaptés aux besoins de la fédération, la phase initiale de la définition d'une fédération est achevée.

Dans l'approche que nous avons choisie, les outils sont définis en deux temps :

1. la définition intrinsèque de chaque outil ;
2. la définition contextuelle de chaque outil.

Pour définir le niveau intrinsèque, nous voulons décrire chaque outil de façon générique, c'est-à-dire en dehors de tout contexte d'application. Pour ce faire, nous introduisons le concept de *modèle d'outil*. Il s'agit d'une description générique qui n'a aucune réalité au sens où le modèle d'outil ne décrit pas un outil installé mais un outil "prêt à l'emploi" (prêt à être installé sur un ordinateur). Par exemple, pour l'outil RCS, outil disponible dans le commerce, son modèle d'outil contient un ensemble de caractéristiques obtenues et connues en lisant les manuels d'utilisation, dossier de spécifications et notices techniques : cela permet de recenser les services qu'offrent RCS et sous quelle forme (commande en ligne, service synchrone ou asynchrone, etc.), les possibles interactions avec son environnement (modification du système de fichier, fenêtres de dialogues avec l'utilisateur, etc.), ...

Nous pouvons établir une analogie avec le concept de *modèle de voiture* (par exemple) qui n'est qu'une vue de l'esprit, qui n'a aucune concrétisation matérielle. Une voiture (véhicule concret) n'étant qu'une instance d'un modèle avec quelques spécificités.

Il est important de noter que le modèle est décrit sur la base (1) des notices et des documents fournis avec l'outil, (2) les spécifications de ce dernier si elles sont disponibles, (3) les recommandations d'usage de l'outil (plate-forme d'exécution, mémoire requise, etc.) ; (4) l'expérience de l'utilisateur n'étant qu'un atout supplémentaire.

La **description intrinsèque** porte sur différentes caractéristiques comme :

- l'ensemble des services offerts par l'outil et classés selon leur type (ligne de commande, message, opération, etc.) ;
- les exécutables de l'outil et leurs chemins d'accès ;
- la possibilité ou non pour l'outil de fonctionner selon le mode client-serveur ;

- le fait d'être portable (décrit la possibilité pour les exécutables de s'exécuter sur différentes plate-formes[57]) ;
- le fait d'avoir ou non un état persistant ;
- le fait de modifier ou non le système de fichiers ;
- le fait de générer des évènements ;
- le fait d'interagir avec l'utilisateur ;
- etc.

Dans le cadre de nos expérimentations, seuls l'ensemble des services fournis, les exécutables et leurs chemins d'accès sont exploités. Les autres informations ne sont pas prises en compte. Cependant, ces caractéristiques sont importantes à prendre en compte pour la génération des façades et l'implémentation des mécanismes d'observation de ces dernières.

Tous les modèles d'outils ainsi définis sont stockés dans une bibliothèque mise à la disposition du concepteur de la fédération. Les définitions des modèles d'outils (incluant l'ensemble des caractéristiques énumérées précédemment) sont décrites/stockées à l'intérieur de classes Java (une classe Java correspondant à une instance unique). Des outils graphiques sont en cours de développement pour permettre à l'utilisateur de rentrer ces informations manuellement et font partie de l'univers commun de contrôle (UCC).

La phase suivante consiste, pour le concepteur de la fédération, à définir les outils "réels" de la fédération, c'est-à-dire, les instances des modèles d'outils sélectionnés ainsi que la topologie de la fédération. Pour cela, le concepteur va, pour chacun des outils (des instances) :

- sélectionner le modèle d'outil dont l'instance est issue ;
- définir la machine sur laquelle est installé l'outil ;
- définir certaines caractéristiques liées à l'installation (au déploiement) de l'outil sur la machine (chemins et répertoires d'accès aux exécutables, etc.) ;
- etc.

Ces différentes caractéristiques sont renseignées au travers d'une interface graphique où le concepteur dispose déjà de la liste des modèles de chaque outil.

En ce qui concerne la **description contextuelle**, il s'agit pour l'utilisateur, de déclarer pour chacune des instances précédemment définies :

- quels sont les rôles endossés par chacune d'elles ;
- l'association ou non à un représentant (voir section IV.4.1).

Cette phase concerne essentiellement l'attribution des rôles définissant ainsi le comportement que suivront chacune des instances lors de l'exécution de la fédération et les règles et contraintes auxquelles elles seront soumises (et déclarées dans la définition des rôles). Ici encore, le concepteur peut utiliser un outil graphique lui permettant d'associer à une instance les rôles qu'elle doit endosser.

La topologie de la fédération

Le fait que chacune des instances soit "localisée" contribue à la définition de la topologie de la fédération. Il est alors possible de savoir à tout instant sur quelle machine se trouve telle ou

[57] C'est le cas, par exemple, des classes Java qui peuvent être interprétées sur plusieurs plate-formes sous réserve de la présence d'une machine virtuelle java.

telle instance. Une localisation est définie au travers d'un certain nombre de caractéristiques (nom complet de la machine avec le domaine ou adresse IP pour identification sur le réseaux, etc.). Actuellement, seules quelques unes des caractéristiques définies sont exploitées. Nous pensons, dans le futur, que d'autres pourraient l'être notamment lorsque nos travaux porteront sur l'étude de l'évolution de la fédération (et, entre autre, savoir quelles sont les machines potentielles de la fédération qui peuvent accueillir telle ou telle instance dans le cas où cette dernière serait mobile, etc.).

V.2.4 Définition des aspects

Les aspects forment le caractère "dictatorial" de la fédération ; en ce sens, plus il y a d'aspects plus le contrôle est "dur" alors que l'inverse traduit un contrôle "mou" (peu ou absence d'aspect).

Le concepteur de la fédération va devoir décider quels sont les services fédéraux qu'il souhaite contrôler et ceux pour lesquels l'autonomie des outils est suffisante. Par analogie, l'autorité gouvernante d'un pays peut définir quels sont les axes (ou thèmes, ou domaines) de la société (du pays) qui doivent être sous contrôle (soumis à l'autorité) et, au contraire, quels sont ceux qui peuvent être laissés à l'initiative des individus. Ce choix ne peut être fait sans avoir une vision globale de la fédération. D'un certain point de vue, le fait de vouloir mettre sous contrôle traduit (1) le degré de confiance que le concepteur va avoir dans les outils (et leurs environnements) qui sont mis à sa disposition et (2) le contrôle sur l'enchaînement des actions des outils (la coordination).

C'est dans le cadre des aspects que le concepteur définit si certaines actions doivent être englobées dans une transaction (sous réserve que cela soit effectivement réalisable au niveau des outils en question). Prenons un exemple.

Imaginons que le Rôle_1 soit chargé d'exécuter les actions A et C et que le Rôle_2 soit chargé d'exécuter l'action B. La suite des actions A, B, C forme une action globale cohérente. Dans le cas d'une fédération "anarchique", sur réception de notifications produites à la suite d'une modification de l'univers commun, les outils endossant les rôles Rôle_1 et Rôle_2 peuvent réagir librement, c'est-à-dire, exécuter les actions A, C et B de façon indépendante, non coordonnée. Si le concepteur veut impérativement garantir que les actions A, B et C se produiront dans cet ordre, il va devoir définir un aspect qui englobe les actions A, B, C selon l'ordre :

```
Rôle_1.A ;
Rôle_2.B ;
Rôle_1.C ;
```

L'ordonnancement est particulièrement important dans le cas où l'action sollicitée auprès d'un rôle retourne un résultat qui est exploité dans la ou les actions ultérieures. Le fait d'envelopper cette suite d'actions dans une transaction traduit le fait que soit l'ensemble de ces actions est réalisé complètement et dans l'ordre proposé, soit aucune d'entre elles ne doit être validée. Par exemple, si l'action C ne peut être exécutée alors que les actions A et B l'ont déjà été et dans cet ordre alors, l'action B effectuée par Rôle_2 devra être défaite ainsi que l'action A effectuée par Rôle_1.

Les aspects intègrent donc la coordination des différents rôles pour atteindre les objectifs fixés par chaque service fédéral.

Ainsi, le concepteur de la fédération doit définir (1) quels sont les services fédéraux qu'il faut contrôler (2) quels sont les aspects à associer à chacun des services fédéraux et (3) pour

chacun d'eux quels sont les rôles impliqués et coordonner les invocations des différents rôles impliqués.

Au niveau pratique les aspects ont été définis à l'aide du langage Java permettant de bénéficier des structures de contrôle "classiques". Plusieurs langages de coordination déjà existants pourraient s'avérer être d'excellents candidats pour exprimer les aspects (par exemple CLF [Andreoli et al. 1999], etc.). Le choix des formalismes pour l'expression des fédérations fera l'objet de travaux ultérieurs.

Dans le cadre des scénarios mis en place, l'ensemble des aspects sont définis à l'intérieur d'une classe Java. Par soucis de simplicité, un aspect est représenté par une méthode de cette classe dont la signature est proche de la signature du service fédéral auquel il est associé. Par exemple, un aspect associé au service fédéral "`terminateActivity(Product p, String exitstate)`" sera défini par la méthode dont la signature est "`terminateActivity_A1(Product p, String exitstate)`".

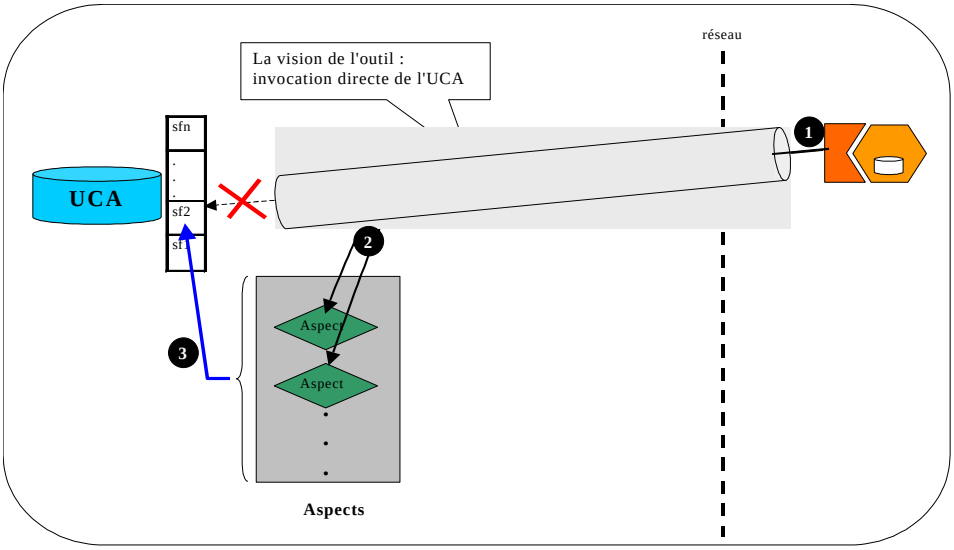

Figure V.3 Déclenchement des aspects

Sur l'invocation d'un service fédéral (Figure V.3 – (1)), tous les aspects associés à ce même service sont exécutés en parallèle (Figure V.3 – (2)). Pour cela, l'ensemble des aspects associé à un même service fédéral est identifié par un groupe de processus Java (classe `java.lang.ThreadGroup`) et chacun des aspects est lui-même un processus (`java.lang.Thread`).

Pour une invocation d'un service fédéral, dès qu'un aspect de l'ensemble des "associés[58]" a terminé son exécution (un code de fin d'exécution est envoyé), l'univers commun de l'application va être modifié : le service fédéral est alors réellement exécuté sur l'univers commun[59] (Figure V.3 – (3)).

V.2.5 Définition des représentants (proxy)

Il est possible, pour chaque instance (outil) d'associer ou non un représentant. Le fait d'associer un représentant permet de traiter les problèmes de conversions (syntaxiques et/ou sémantiques) en amont de l'outil. Dans le cas contraire (pas de représentant), ces conversions

[58] C'est-à-dire l'ensemble des aspects associés au même service fédéral.
[59] Il s'agit d'un choix relativement arbitraire. Le cas se justifie s'il n'existe qu'un aspect associé à un service fédéral et que ce dernier est transactionnel.

sont à la charge de la façade : ainsi, cette dernière perd dans ce cas son indépendance par rapport à une fédération particulière.

Le représentant, de par sa position au sein de l'architecture et de par ses "missions", constitue le lien entre chaque outil et une fédération spécifique (un UCA spécifique).

Pour chacun des outils de la fédération, le concepteur doit définir l'implémentation de son représentant tel qu'il effectue l'ensemble des opérations de conversions nécessaires. Dans l'exemple ci-dessous, sur invocation de la méthode `void transfert(Product p)`, le représentant, à partir de l'objet *p* de type "produit" (concept de l'UCA) en déduit un objet *file* de type "fichier" (concept des outils RCS et CVS).

```
public class RCSProxy extends Proxy {

...
 public void transfert(Product p) {
        String name = p.getName();
        byte[] file = p.getBytes();
        java.io.File output = new java.io.File(name);
        FileOutputStream streamOUTfileOUT;
        try {
               streamOUTfileOUT = new FileOutputStream(output);
               try {
                      streamOUTfileOUT.write(file); //writes the file
               }
               catch (...) {...}
        }
        catch (...) {...}
        ((RCSWrapper)this.wrapper()).checkIn(name);
 }
...
}
```

Implémentation en Java du représentant de l'outil RCS (extrait)

La nature même du représentant fait qu'il nous paraît difficilement concevable de pouvoir automatiser son obtention, comme l'ont déjà remarqué quelques auteurs [Busse et al. 1999]. Celle-ci reste le fruit d'une activité intellectuelle et manuelle de l'être humain pour recenser et définir les conversions nécessaires.

Cependant, la définition des instances va permettre d'aider le concepteur à recenser les conversions nécessaires. En effet, la description contextuelle de chaque outil (voir section V.2.3) permet de connaître pour chaque instance :

- les services fournis par elle : soit *S(i)* cet ensemble ;
- les rôles endossés par elle et pour chacun de ces rôles les services abstraits à fournir : soit *R(i)* cet ensemble.

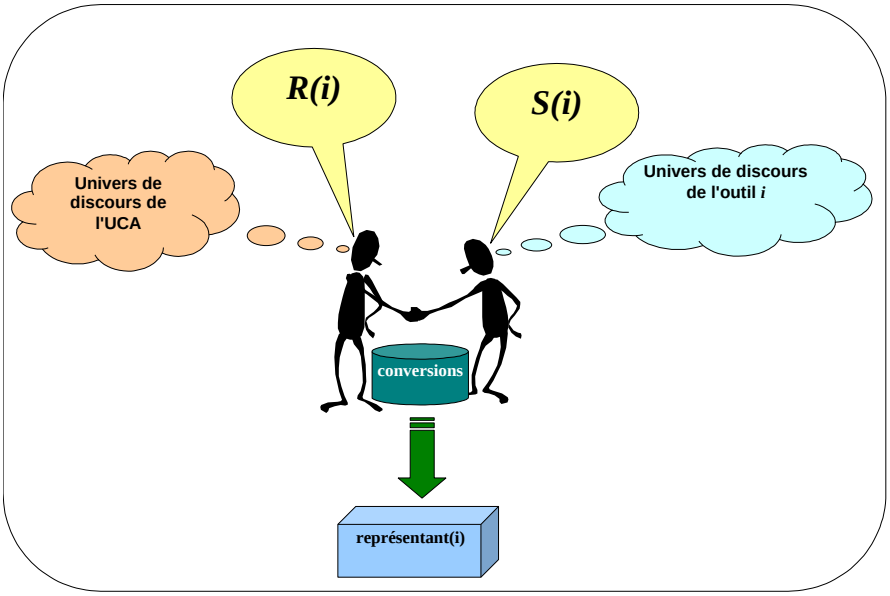

Figure V.4 Principe de définition du représentant d'une instance *i*

La définition du représentant décrit normalement une application *R(i)* → *S(i)* (voir Figure V.4) : à tout élément de *R(i)* est associé un et un seul élément de *S(i)*. Chacune de ces associations se fait manuellement, par le concepteur, en faisant appel à des procédures de conversions déjà disponibles lorsque cela est possible.

Les représentants vont implémenter les mécanismes de communication de la fédération qui seront utilisés avec les façades des outils. Entre autre, ce sont les représentants qui sont considérés être les souscripteurs et qui reçoivent les notifications ; ils ont donc la charge de répercuter les notifications auprès des façades ce qui permet de s'adapter au mode d'interaction supporté par l'outil (communication synchrone ou/et asynchrone, point-à-point, etc.).

V.2.6 Définition des façades (wrapper)

Pour chacun des outils de la fédération, le concepteur doit définir l'implémentation de sa façade. Une façade est définie, dans le cadre du prototype, dans une classe Java. A chaque service de l'outil (dont la description est incluse dans le modèle de l'outil), l'utilisateur va associer une opération (méthode) enveloppant le service[60]. Cette description de façade se base sur des informations et des caractéristiques fournies dans le modèle de l'outil (chemins et nom complet des exécutables, etc. – voir section V.2.3).

L'exemple suivant (voir code page suivante) montre, à partir de l'invocation de la méthode `void checkIn(String f)` comment cette dernière enveloppe la commande en ligne permettant effectivement de faire faire une révision à l'outil RCS (du fichier dont le chemin complet est contenu dans la chaîne de caractères *f*). En particulier, la méthode `doCmd(...)`, définie dans la classe mère `Wrapper.java` (non présentée ici), effectue les appels nécessaires au système d'exploitation permettant d'exécuter des commandes en ligne. Les mécanismes

[60] Cela est fonction de la classification des services de l'outil. Dans le cas où le service est un service "prêt-à l'emploi", l'encapsulation n'est pas forcément nécessaire.

d'encapsulation génériques[61] permettant l'invocation des services des outils sont définis dans la classe mère à partir de laquelle sont définies les façades de chaque outil.

```
public class RCSWrapper extends Wrapper {

 ...
 public void create(String f) {
       System.out.println("=== The RCS wrapper is calling the RCS tool for
                                service request ...");
       boolean execute = doCmd(1,"c:\\Program Files\\ComponentSoftware\\CS-
                                RCS\\System\\csrcs Create "+f);
 }
 public void checkIn(String f) {
       System.out.println("=== The RCS wrapper is calling the RCS tool for
                                service request ...");
       boolean execute = doCmd(1,"c:\\Program Files\\ComponentSoftware\\CS-
                                RCS\\System\\csrcs CheckIn "+f);
 }
...
}
```

Implémentation en Java de la façade de l'outil RCS (extrait)

Les méthodes définies dans les façades représentent, pour bon nombre d'entre elles (en particulier les méthodes "publiques") les méthodes d'accès à distance de la façade : il s'agit donc de l'ensemble des méthodes enveloppant les services de l'outil (associé à la façade en question).

[I] Certains éléments de la définition (implémentation) de la façade figurant ci-dessus pourraient être générés automatiquement. En effet, nous avons dans le modèle de l'outil RCS, l'information que le service de révision "checkIn" est :

- une commande en ligne,
- dont la syntaxe est `"c:\\Program Files\\ComponentSoftware\\CS-RCS\\System\\csrcs CheckIn"`,
- suivie du nom du fichier à réviser en paramètre,
- sur une plate-forme MS-Windows.

[II] D'autre part, nous avons défini un certain nombre de services génériques dans la classe mère `Wrapper` et, entre autres, une méthode `doCmd(...)` qui :

- invoque un service qui est une "ligne de commande" en appelant l'interpréteur de commande[62] de la plate-forme ;
- permet de configurer certaines options comme la persistance ou non de la fenêtre effectuant l'appel à l'interpréteur de commande, après l'invocation (options –c ou –k, etc.) ;
- pour la plate-forme MS-Windows.

[61] L'implémentation des services génériques sont fonction de la plate-forme (MS-Windows, Unix, ...), etc. Cette information peut être automatiquement exploitée puisque la plate-forme sur laquelle s'exécute l'outil est une des caractéristiques de la description contextuelle de l'outil (voir section V.2.3).
[62] Couramment appelé "shell"

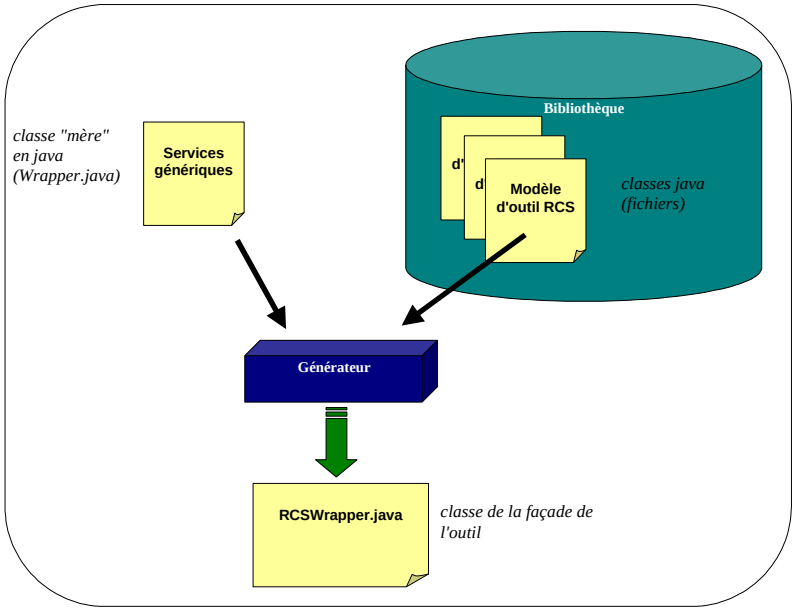

Figure V.5 Principe d'obtention des façades

Compte tenu des informations dont le concepteur dispose [I] et [II], un générateur de façades[63] pourrait directement en se basant sur ces informations, construire une partie de la classe `RCSWrapper` et générer, par exemple, la méthode `void checkIN(String f)` suivant le processus de génération de la figure V.5 :

```
 public void checkIn(String f) {
       boolean execute = doCmd(1,"c:\\Program Files\\ComponentSoftware\\CS-
RCS\\System\\csrcs CheckIn "+f);
 }
```

Cette méthode exécute l'équivalent d'un appel effectué par l'utilisateur tel que celui représenté par la figure V.6.

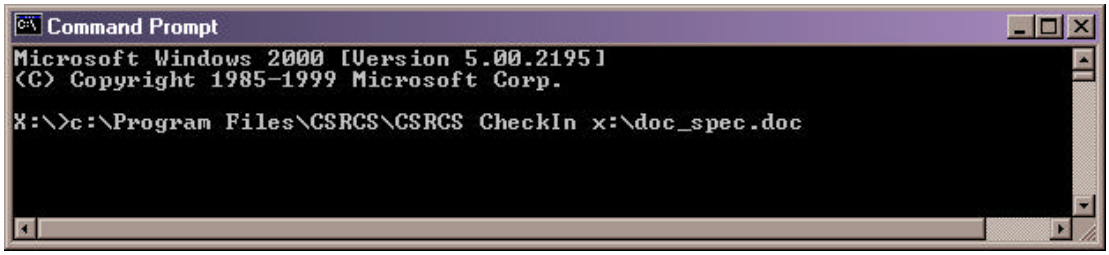

Figure V.6 Invocation d'une révision en ligne de commande

Cet exemple montre l'exploitation que l'on peut faire des informations définies dans le modèle de fédération (partie de l'univers commun de contrôle).

[63] Nos travaux n'ont pas exploré davantage la question de l'aide à la génération des façades. Un tel générateur (logiciel) n'existe pas à l'heure actuelle.

V.2.7 Obtention de la fondation de contrôle à partir du modèle de fédération

La définition d'une fédération d'outils, telle que nous l'avons implémentée, repose sur la définition d'informations stockées dans des classes Java. Ces classes et interfaces Java couvrent :

1. la définition des *services fédéraux* de l'UCA ;
2. la définition des *rôles* ;
3. la définition des *outils* et *instances* (et l'attribution des rôles) ;
4. la définition des *aspects* ;
5. la définition des *représentants* ;
6. la définition des *façades*.

Ces classes Java représentent donc le modèle de la fédération dont le modèle de contrôle fait partie.

La définition des *représentants* se sert :

- de la définition des *outils* et *instances* et de la définition des *rôles* endossés par chacun d'eux.

La définition des *façades* se sert :

- de la définition des *instances* basées sur les modèles d'outils.

Un ensemble de classes a été implémenté servant de support aux classes contenant la définition du modèle de fédération (c'est notamment le cas des classes et des services "génériques"). Ces classes sont indépendantes d'une fédération particulière.

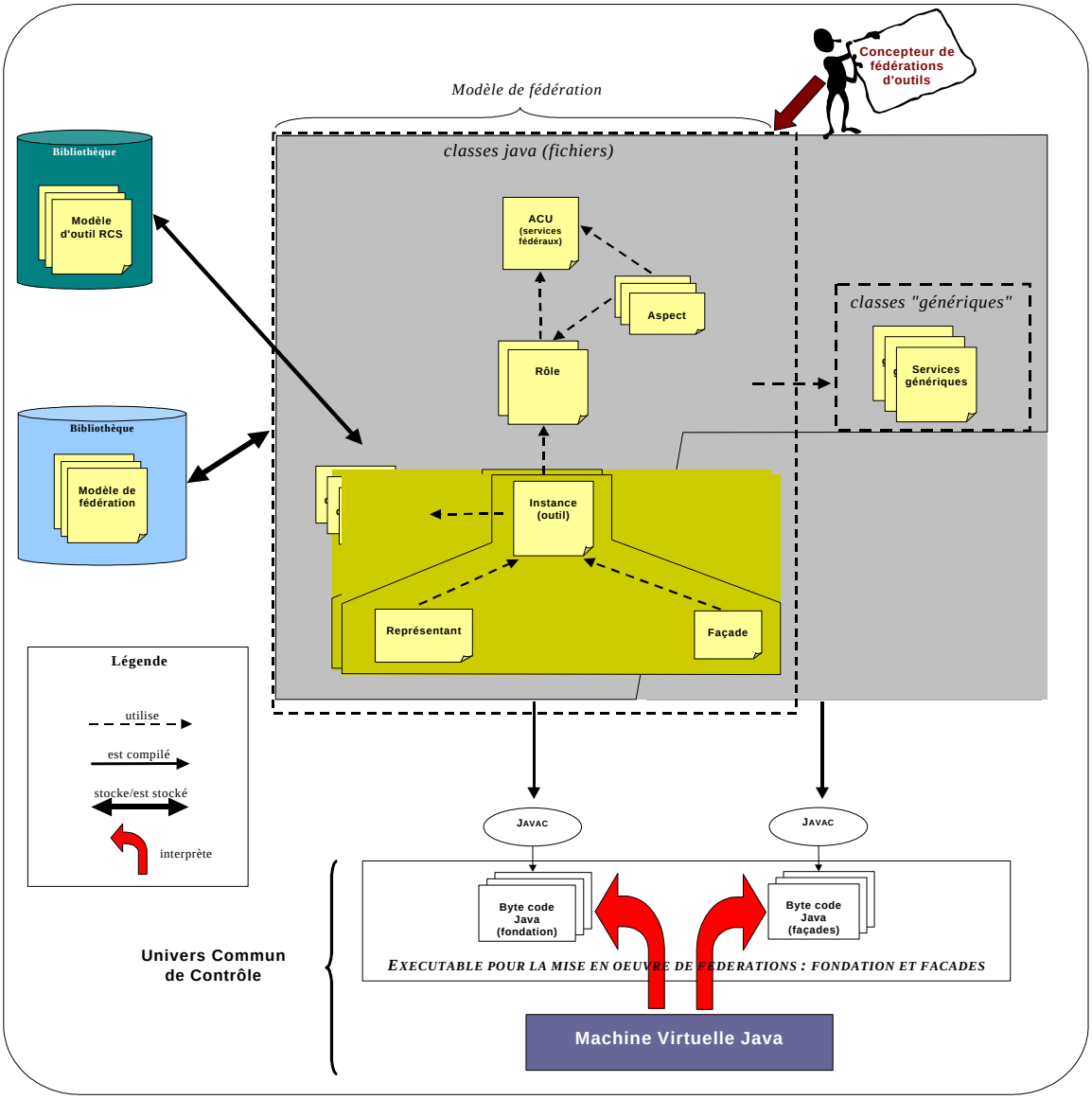

Figure V.7 Schéma de principe de mise en oeuvre des fédérations d'outils

Les classes définissant le modèle de fédération servent de support (voir Figure V.7) :

- à la fondation de contrôle ;
- aux façades de chaque outil.

Elles sont, à l'exécution, interprétées par la ou les machines virtuelles Java. L'état des objets, instances (au sens de la programmation orientée objet) de ces classes, stockés dans la machine virtuelle qui interprète le byte code de la fondation de contrôle, ainsi que le byte code forment l'état de ce que nous avons appelé l'*univers commun de contrôle*. Un participant de la fédération est déterminé par :

- la définition de l'instance ;
- la définition de son représentant (si un représentant lui est associé) ;
- la définition de sa façade.

Les classes Java des façades se trouvent sur les ordinateurs sur lesquels s'exécutent les outils (instances). Les autres classes se trouvent centralisées sur la machine sur laquelle se trouve l'univers commun de l'application.

V.2.8 Gestion de la confidentialité, de la sécurité

Un des aspects importants des systèmes distribués – donc des fédérations d'outils, est la gestion de la confidentialité et de la sécurité... surtout dans une perspective inter-organisationnelle (consortiums d'entreprises, etc.).

Prenons un exemple. Dans le cadre du scénario de transfert de produit, l'outil RCS s'exécute quelque part, sur une machine physique (un ordinateur) appartenant à une organisation. Le modèle de l'outil RCS comprend la définition du service "History" (entre autres services) permettant d'avoir l'historique d'un fichier géré (révisé) par RCS. Par contre, ce service peut, dans certains cas, être considéré comme un service "risqué", c'est-à-dire et dans l'exemple, comme pouvant violer les règles de sécurité et de confidentialité, surtout s'il peut être invoqué à distance. En effet, le résultat d'une demande d'historique sur un fichier peut revêtir un caractère de confidentialité.

Dans le cas général, la sécurité des COTS est assurée par leurs façades [DSoS 2000, OMG 2000] qui ne contiennent que la description des services pertinents et conformes aux règles de sécurité de l'organisation qui les développe. Cela reste valable dans le cadre des fédérations d'outils, notamment parce qu'il est impossible au niveau de la fondation de contrôle (organe de contrôle centralisé), de contrôler ou de garantir l'implémentation effective des façades (qui sont sous le contrôle des organisations participant à la fédération)[64] ; d'autre part, nous avons fait l'hypothèse d'effectuer le contrôle de la fédération en amont des outils, permettant ainsi d'avoir des façades "génériques"[65].

Il s'agit donc pour le concepteur qui doit définir le modèle de fédération, de tenir compte des contraintes de sécurité. Dans le cadre de l'exemple avec RCS, pour que le service "History" ne puisse être invoqué dans le cadre de la fédération, il suffirait que l'ensemble des rôles endossés par l'outil en question n'invoquent pas le service "History". C'est ce qui se passe dans le scénario où le rôle *WorkspaceManager* ne fournit pas de service qui implique le service "History" : la description du représentant de l'outil RCS n'effectue aucun appel du service "History". Ces "restrictions" couvre les invocations de services dans le cadre des fédérations d'outils telles que nous avons définies. En revanche, rien ne vient empêcher un utilisateur d'invoquer le service "history" en tapant la ligne de commande appropriée. Cette question nous ramène à la discussion portant sur la gestion de l'autonomie locale des outils abordée au cours du chapitre précédent (section IV.4.3) et reprise dans les perspectives de ce travail de thèse.

Dans les systèmes à base de COTS, la définition des rôles est relativement pragmatique dans le sens où elle tient compte des outils réellement disponibles (voir section IV.6.3). Une fois la définition des rôles effectuée, le concepteur devra, au besoin, modifier certains rôles afin de rendre indisponibles certains services, ou définir d'autres rôles en rapport avec les contraintes de sécurité et de confidentialité.

La sécurité est également traitée avec le concept d' *interdictions* portant sur les rôles (voir sections IV.3.2 et V.2.2).

La gestion de la confidentialité et de la sécurité, à laquelle sont confrontées les entreprises n'est pas explicitement traitée dans le cadre de cette thèse. Certains concepts présentés dans ce

[64] Rappelons-nous que le concepteur de la fédération, n'a, dans le cas général, pas ou peu de contrôle sur les outils qui sont mis à la disposition de la fédération par les organisations du consortium.

[65] Nous avons également réfléchi sur l'opportunité d'avoir des façades faites "sur mesure", c'est-à-dire conformes aux spécifications d'une fédération particulière (non générique), en incluant le contrôle au niveau des façades. Cependant, dans le cas général, nous ne pourrions garantir que les façades ne soient pas détournées de leurs prérogatives initiales (ce qui revient au point précédent).

document permettent d'aborder cette problématique sans pour autant apporter une réponse générale et exhaustive. Cette question constitue une des perspectives de nos travaux.

V.3 Construire des fédérations : cas d'un scénario

Différentes "politiques" de contrôle peuvent être mises en place, à la fois en fonction de la nature de la problématique, de l'application mais également selon le mode de fonctionnement le plus adapté à la situation (localisation, distribution des outils, degré d'autonomie et nécessité du contrôle, etc.). Le choix appartient au concepteur de la fédération qui doit cependant faire face aux contraintes imposées par les participants de la fédération, notamment par le fait que les outils, par hypothèse autonomes, sont immergés dans un contexte (l'entreprise) qui ne peut pas être maîtrisé, sur lesquels il n'est pas possible d'intervenir.

Nous allons regarder, dans le cas du scénario du transfert de produit, et en fonction de la politique choisie, l'incidence de cette dernière sur le mode de fonctionnement de la fédération. Nous constaterons que ce mode de fonctionnement est déterminant pour comprendre les fédérations d'outils et l'importance de ces dernières dans les projets et les applications à mener, notamment dans le fait de pouvoir diriger l'exécution de la fédération avec plus ou moins "d'autorité".

V.3.1 Les éléments de la fédération

Nous reprenons la définition de la fédération (voir section V.5.1) basée sur un ensemble constitué :

- de la fondation de l'application,
- de la fondation du processus,
- de la fondation de contrôle,

ces deux dernières étant optionnelles.

La fondation de l'application

Pour la fondation de l'application, nous devons définir un univers commun, l'UCA, un ensemble d'outils pour réaliser l'application et un ensemble de mécanismes d'interaction.

L'application à réaliser étant un processus, l'UCA regroupe l'ensemble des concepts et des entités basées sur ces concepts permettant de définir et de mettre en oeuvre un processus. Nous retrouvons les concepts, les formalismes et les objets permettant d'exprimer un modèle de processus et l'état des instances de ce modèle (voir les travaux permettant d'exprimer un processus logiciel dans le second chapitre). Dans la cas de l'exemple, nous utilisons l'environnement et le langage APEL. L'UCA est donc la base de données contenant la description et l'état commun du processus à exécuter [Amiour 1999].

Les outils de la fédération sont les outils impliqués dans l'application, c'est-à-dire le processus dont la modélisation comporte deux activités avec entre elles un transfert d'un document de spécification.

Cet ensemble d'outils est constitué (voir Figure V.8) :

- des composants de l'EGLCP APEL (nous nous intéresserons particulièrement au moteur de processus) ;

- d'un gestionnaire des espaces de travail (Adele) ;
- de deux gestionnaires de versions (RCS et CVS) ;
- d'autres outils logiciels (MS-Word, etc.).

Les mécanismes d'interaction comprennent un bus logiciel d'objets distribués permettant l'invocation d'objets à distance Java/RMI ainsi qu'un serveur de message basé sur les spécifications JMS.

V.3.2 Le fonctionnement de la fédération

On part du mode de fonctionnement basé sur le paradigme de "l'anarchie" selon lequel les outils peuvent "librement" modifier et observer l'univers commun (de l'application) [Estublier et al. 1998b, Estublier et Verjus 1999] et peuvent librement participer ou non à la fédération. Selon ce mode, aucun contrôle n'est pratiqué. Comme nous l'avons vu, ce mode de fonctionnement présente l'inconvénient majeur de ne définir aucune coordination entre les outils. Ainsi, n'importe quel outil de la fédération peut forcer la terminaison de l'activité source (activité de spécifications) et ainsi, provoquer le transfert du document (produit) vers l'activité cible (activité de revue). Or, la terminaison de l'activité et le transfert du produit sont des services fédéraux de l'application (l'application est dans ce cas un processus logiciel) que l'on souhaiterait *réserver* à l'outil moteur de processus. Dans ce cas, il faut décrire un rôle *ProcessEngine* (moteur de processus) qui seul peut (aura le droit) invoquer/utiliser le service fédéral (de l'application) "`terminateActivity(Activity act, String exitstate)`". L'utilisation de ce service fédéral est strictement réservée à ce rôle (les autres rôles décrits dans la fédération ne peuvent prétendre l'utiliser).

Par contre, d'autres outils de la fédération , même s'ils n'initient pas le transfert, n'en sont pas moins impliqués/concernés dans l'acte de transférer : notamment les outils de gestion de versions, le gestionnaire d'espaces de travail, etc. Nous pouvons considérer deux alternatives permettant de satisfaire aux besoins :

1. envoyer des notifications (après abonnements) aux outils impliqués ;
2. invoquer directement les services pertinents de chaque outil en regard des tâches à accomplir.

La différence sémantique entre les deux alternatives est relativement importante. Dans la première alternative, il s'agit, pour les outils d'être avertis lors de la réalisation du service fédéral (ils sont *concernés*) ; mais la réalisation effective du service fédéral ne dépend pas des outils dans le sens où le service fédéral sera exécuté quel que soit le comportement de chaque outil. Dans la seconde alternative, les outils sont réellement *impliqués* dans le service fédéral et la réalisation effective de ce service (dans la réalité) dépend totalement des outils dans le sens où le comportement individuel de chaque outil va être pris en compte pour l'exécution du service fédéral.

V.3.3 Autonomie des outils

La première alternative est la moins contraignante. En effet, les outils (ou leurs façades) sont "libres" à la réception de l'évènement de réagir (ou non). Les notifications étant, par définition, des sollicitations asynchrones, la fédération peut continuer à s'exécuter sans tenir compte du comportement individuel de chacun des outils.

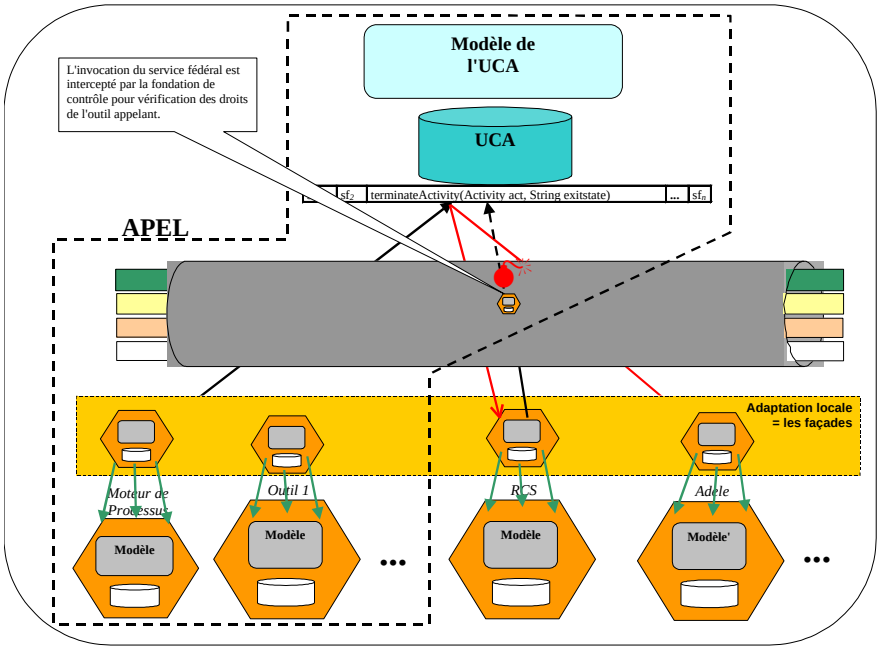

Figure V.8 Contrôle de l'invocation d'un service fédéral de l'UCA

Selon ce mode de fonctionnement, les invocations des services fédéraux de l'univers commun sont contrôlées alors que l'observation de l'état de ce même univers commun n'est pas contrainte, pas plus que ne le sont les comportements locaux des différents outils.

Nous ajoutons à la fondation de l'application une fondation de processus et une fondation de contrôle.

La fondation du processus

Le cas particulier de ce scénario où l'application est un processus nous amène à montrer l'identité entre la fondation de l'application et la fondation du processus FP = FA.

La fondation de contrôle

Dans la mesure où le mode de fonctionnement suit le paradigme de l'anarchie, nous avons montré que la fondation de contrôle n'était pas indispensable. Cependant sa présence peut se justifier si nous souhaitons définir la fédération (les outils et leurs caractéristiques).

V.3.4 Vers une meilleure synergie entre les outils

En reprenant le cas précédent, nous pourrions décrire explicitement le comportement que doivent avoir les outils pour réaliser "réellement" le service fédéral. Pour préciser et "dicter" le comportement des outils, nous définissons un aspect associé au service fédéral "`terminateActivity(Activity act, String exitstate)`". Cet aspect va invoquer les services synchrones des différents outils impliqués, à savoir :

- le gestionnaire des espaces de travail pour qu'il mette à jour les espaces de travail des utilisateurs impliqués dans les activités source et cible ; cela se traduit par la suppression de l'objet document de l'espace de travail de l'utilisateur chargé de l'activité cible et par la création de l'objet document dans l'espace de travail de l'utilisateur chargé de l'activité cible ;
- le gestionnaire de versions pour qu'il effectue une révision du document transféré.

Puis, le service fédéral sera invoqué sur l'univers commun permettant de refléter (mettre à jour) les actions effectuées localement par les outils au niveau de l'univers commun de l'application.

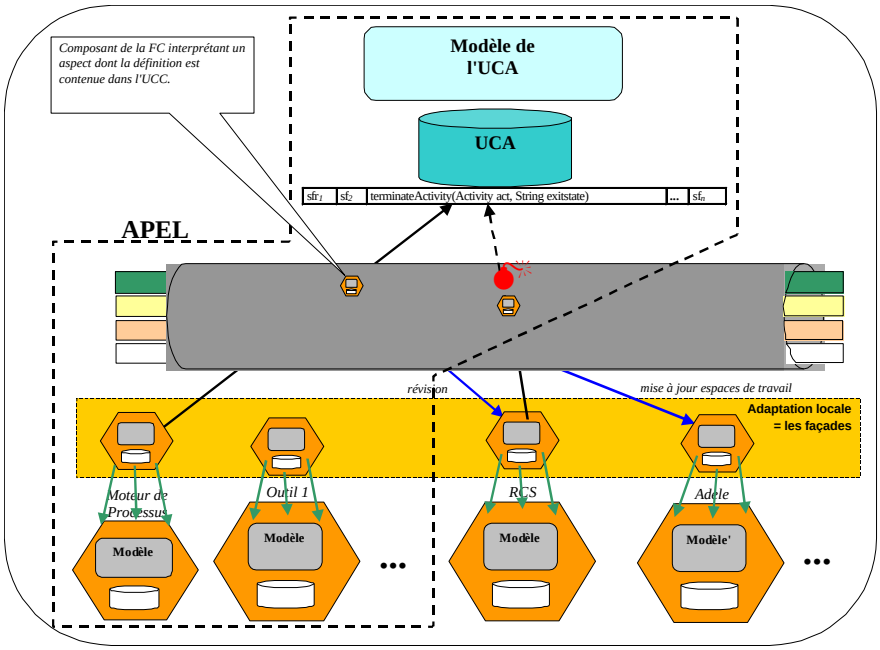

Figure V.9 Réalisation d'un aspect associé à un service fédéral de l'UCA

La fondation de contrôle

Contrairement à l'alternative précédente où le contrôle n'était absent, nous définissons un aspect associé au service fédéral "`terminateActivity(Activity act, String exitstate)`". Cet aspect (voir Figure V.9) contient l'ensemble des rôles impliqués, la coordination de ces derniers ainsi que les propriétés à appliquer (gestions des transactions et des contraintes temporelles). Nous avons donc un modèle de fédération plus complet que le précédent et un ensemble d'aspects constituant la "couche" coordination de la fondation de contrôle.

V.3.5 Conclusion

Nous remarquons que suivant le type de politique implanté (anarchie, dictature, etc.), l'initiative des outils est plus ou moins grande tout comme l'influence de la fondation de contrôle. Par analogie avec une fédération d'Etats, et au travers de l'exemple, nous montrons que la fondation de contrôle joue le rôle d'un gouvernement fédéral plus ou moins directif (autoritaire).

Plus la description de la fédération est complète (par le modèle de fédération) plus la fondation de contrôle a de l'importance dans le fonctionnement de la fédération.

V.4 Prototype et Expérimentations

Nous pensons que les expérimentations dans le cadre des fédérations d'outils, sont particulièrement importantes pour vérifier la véracité de nos propos et valider notre approche. Dès le développement du prototype, des scénarios de tests ont été mis en place permettant de simuler le comportement des fédérations. Puis, sur la base du scénario de transfert de produit,

plusieurs cas ont été mis en oeuvre, en fonction du comportement de la fédération désiré (mode plus ou moins "anarchique" ou "dictatorial", etc.).

V.4.1 Expérimentations

En ce qui concerne l'étude de cas, nous avons choisi le scénario de transfert de produit (voir quatrième chapitre et section V.3) dont la représentation graphique, effectuée à l'aide de l'éditeur d'APEL v5 est présentée par la figure V.10.

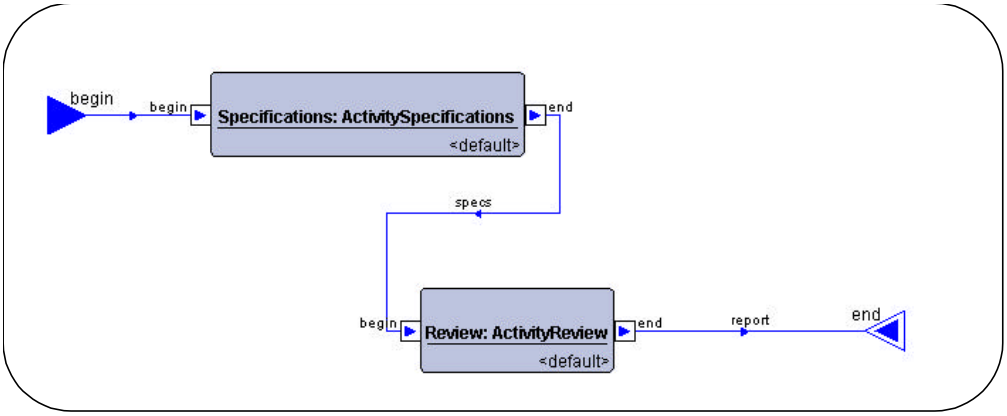

Figure V.10 Scénario du transfert de produit (formalisme graphique d'APEL v5)

Pour ce scénario nous avons eu besoin des outils suivants (voir figures V.8 et V.9) :

- un environnement de génie logiciel centré processus sachant que l'application est un processus (simple) ;
- deux gestionnaires de versions (RCS et CVS).

La première version du prototype implémentait :

- une fondation de contrôle basée sur les concepts de rôles (et donc des droits et devoirs) et d'aspects (définissant la coordination des différents rôles) ;
- un représentant et une façade pour l'outil RCS ;
- un représentant et une façade pour l'outil CVS ;
- un modèle de contrôle défini dans les classes Java ;
- un moteur de processus ad-hoc ;
- l'utilisation d'un sous ensemble des services fédéraux[66] présentés dans [Francou 2001, Beydeda et Cîmpan 2001].

Le modèle d'une telle fédération se trouve, à la fois dans les classes Java du prototype qui se trouvent dans l'annexe B, et sous forme d'une description utilisant le langage FTML (voir section V.5).

V.4.1.1 Limitations dans la réalisation des expérimentations

Le prototype présentait cependant les limitations suivantes :

- non prise en compte de la distribution des outils ;
- modèle de fédération non persistant ;

[66] Services recensés pour les besoins de la fédération de PIE [Beydeda et Cîmpan 2001].

- non prise en compte des transactions et transactions distribuées ;
- non prise en compte de l'EGLCP APEL : ce dernier étant remplacé par un composant moteur de processus (PE), codé de façon ad-hoc pour les besoins du scénario.

La fondation de contrôle a été implémentée sans avoir recours à une base de données pour l'UCC ; ce dernier est contenu dans les classes Java de la fondation de contrôle. Ainsi, l'UCC ne bénéficiait pas d'un état persistant, ce dernier étant constitué par l'état des différentes variables de l'implémentation (code) de la fondation de contrôle.

Quant à l'univers commun du processus (UCP), les travaux les plus récents sur la définition d'un modèle commun de processus constituent les bases de la fondation du processus [Amiour et Estublier 1998, Amiour 1999, Estublier et al. 1998b]. Un EGLCP selon l'approche prise dans [Amiour 1999] constitue ce que nous avons appelé dans le quatrième chapitre, la fondation du processus. Dans la mesure où un composant moteur de processus, simple, a remplacé l'EGLCP, nous n'avons pas pu réaliser de tests mettant en oeuvre un univers commun de processus intégrant les différents aspects d'un EGLCP complet.

V.4.1.2 Exécution du scénario – Phase d'expérimentation

Nous allons décrire les différentes étapes du scénario, l'intervention de la fondation de contrôle. Pour cela, nous commenterons les traces générées par le prototype au cours de l'exécution.

L'initialisation des outils et de la fédération est assurée par une classe Java (`Bootstrap.java`), permettant le démarrage de la fondation de contrôle et le départ du scénario.

Sur l'invocation de cette classe :

- l'API de l'univers commun de l'application est activée, mettant à la disposition des outils l'ensemble des services fédéraux,
- la fondation de contrôle est démarrée,
- les façades des outils sont initialisées,
- les objets du scénario (de l'instance du modèle du processus de transfert) sont alors créés,
- le scénario commence à s'exécuter.

```
C:\java\jdk\jdk1.3\bin\java.exe   federation.test.Bootstrap
Working Directory - X:\adele\data\Proto\v02\src\federation\
Class                                  Path                                    -
X:\adele\data\Proto\v02\classes;.;c:\java\ide\Kawa3.5\kawaclasses.zip;

 c:\java\jdk\jdk1.3\lib\tools.jar;c:\java\jdk\jdk1.3\jre\lib\rt.jar;c:\java\jd
k\jdk1.3\jre\lib\i18n.jar


  === The Application Common Universe (ACU) API is now created ...
 === The Right Control Server is started
 === Federation model is being loaded ...
 === The scenario is now reified : all objects have been created in the ACU
 === The Foundation Control is now started ...
 === The APEL Process Engine wrapper is now started
...
```

Initialisation du scénario

A l'exécution, la façade (wrapper) du moteur de processus[67] (une fois le processus du transfert de produit modélisé – voir [Estublier et al. 1998a, Amiour 1999, Dami 1999]) va s'enregistrer auprès de la fondation de contrôle. Cet enregistrement est facultatif mais il permet, pour la fondation de contrôle de savoir si tel ou tel outil a bien été enregistré. Dans le cas contraire, la fondation de contrôle adopte un comportement par défaut défini par le concepteur de cette dernière.

```
...
=== Tool ProcessEngine is now registered to the federation
 === The tool ProcessEngine has the ID tool-001
...
```

Enregistrement d'un outil

L'enregistrement d'un outil auprès de la fédération (la fondation de contrôle) se fait au travers de la méthode `void register(String toolName)`. Cet appel est intercepté par la fondation de contrôle et est redirigé vers l'appel d'une méthode `String register(String toolName)`. Cette dernière retourne, non pas à la façade de l'outil, mais à son "proxy", un identifiant (chaîne de caractères). Tout cela est caché à l'outil et aux utilisateurs de l'outil. La version 1.3 du JDK (Java Development Kit) nous a permis d'utiliser les "proxy"[68] Java (classe `java.lang.reflect.Proxy`) dont l'utilisation dans le cadre du prototype que nous avons développé est illustrée par la figure V.11. L'implémentation de la fondation de contrôle utilise les "proxy" Java pour intercepter les invocations afin de les rediriger et d'effectuer ainsi les procédures de contrôles (vérifications des droits des outils, invocations des aspects, etc.).

A chaque façade est associé un "proxy" qui traite de l'ensemble des invocations qu'effectue la façade. La technologie et la mise en place des "proxy" Java est implémentée dans la classe `Wrapper`, super-classe de l'ensemble des classes définissant les façades de chaque outil. Ainsi, la classe `PEWrapper` hérite de la classe `Wrapper`.

[67] Il s'agit d la classe `PEWrapper`.

[68] A ne pas confondre avec le concept de *représentant* (*proxy* en anglais) introduit dans le quatrième chapitre et repris tout au long du document.

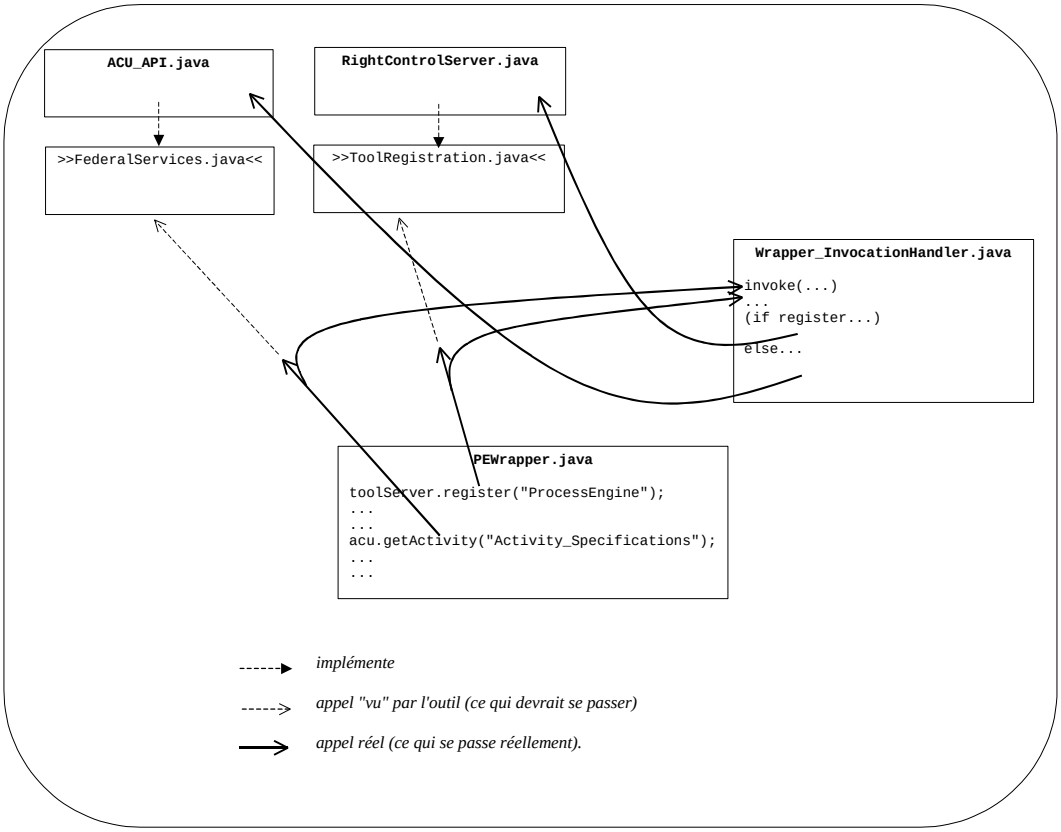

Figure V.11 Utilisation des "proxy" Java pour capturer les invocations des services fédéraux

L'identifiant retourné par la méthode `String register(String toolName)` est alors stocké par le "proxy" Java de la façade. A la suite de quoi, chaque appel effectué par la façade sera automatiquement capturé par la fondation de contrôle, redirigé vers le "proxy" Java de la façade qui répercutera l'appel (voir Figure V.11).

Le "proxy" va vérifier, pour chaque invocation, si cette dernière est autorisée ou non (c'est-à-dire si aucune interdiction n'a été définie sur les rôles endossés par l'outil et le service fédéral invoqué).

Le contrôle se fait en réalité a deux niveaux :

- au niveau des façades des outils ;
- au niveau de la fondation de contrôle.

Le contrôle au niveau des façades est mis en place de façon totalement transparente pour les façades ; toute façade (implémentation Java) doit simplement hériter de la classe `Wrapper`. Une fois l'outil déclaré, grâce à la méthode `String register(String toolName)` (voir précédemment), toutes les invocations émanent d'une façade sont ensuite déroutées vers leurs "proxy" Java respectifs qui effectue le contrôle des droits.

Pourtant, rien n'oblige le développeur d'une façade de se déclarer auprès de la fondation de contrôle. Dans ce cas, l'outil associé à cette façade, ne pouvant être clairement identifié, sera contraint d'adopter un comportement par défaut défini par le concepteur de la fédération. Pour mettre en oeuvre un comportement par défaut, tout appel émanant d'un outil (ou de sa façade) non identifié sera intercepté par le contrôle de second niveau (au niveau de la fondation de contrôle). Ce dernier va intercepter les appels vers l'univers commun et va appliquer une politique par défaut : soit accepter l'invocation même si l'appelant est inconnu, soit la rejeter. Cette politique est définie par le concepteur de la fédération.

Ces deux niveaux permettent, ensemble, de prendre en compte :

- des façades implémentant un mécanisme de "proxy" Java (premier niveau) ;
- des façades "inconnues" (sans avoir recours aux classes de base que nous fournissons - pour lesquelles le premier niveau est inopérant).

Au cours de l'exécution du scénario, la façade du moteur de processus interprète l'instance du modèle de processus décrit par la figure V.10. Le moteur de processus invoque successivement :

- le service fédéral `getActivity("Activity_Specifications")` permettant de récupérer l'activité (instance – objet `spec`) de spécifications (activité source du transfert) ;
- le service fédéral `terminateActivity(spec, "exit")` sur l'univers commun.

```
...
  === PE wrapper invokes the ACU ...
  === The ACU Federal Service terminateActivity invocation has been caught by the Control
Foundation...
...
```

Invocation du service fédéral et intervention de la fondation de contrôle

Cette vérification est effectuée par une instance de la classe `RightControlServer` qui est un processus de la fondation de contrôle (hérite de la classe `java.lang.Thread`). Dans le cas où l'invocation est permise, la fondation de contrôle établit alors la liste des aspects qui sont associés au service fédéral invoqué.

Si aucun aspect n'a été trouvé, l'invocation du service fédéral sur l'univers commun pour modification de son état devient alors effective.

Si, au contraire, un ou plusieurs aspects ont été associés au service fédéral invoqué, la fondation de contrôle lance un processus (héritant de `java.lang.Thread`) qui va traiter l'ensemble de ces aspects (il s'agit d'une instance de la classe `ACU_API_Invocation`). Tout d'abord, un identifiant unique est attribué au groupe d'aspects à exécuter. Chacun de ces aspects est en soi un processus indépendant rattaché au processus gérant le groupe. L'identifiant est géré par la fondation de contrôle qui peut déterminer à tout moment quels sont les aspects en cours d'exécution et à quelle invocation de quel service fédéral ils sont rattachés. Les services fédéraux peuvent être synchrones (invocations de méthodes) ou asynchrones (envois et réception de messages). Aussi, les messages de traces, lors de l'exécution du scénario, font état de tests portant sur le fait que le service fédéral soit une méthode ou un message. Dans le cadre de nos expérimentations, seules les services synchrones ont été soumis aux tests.

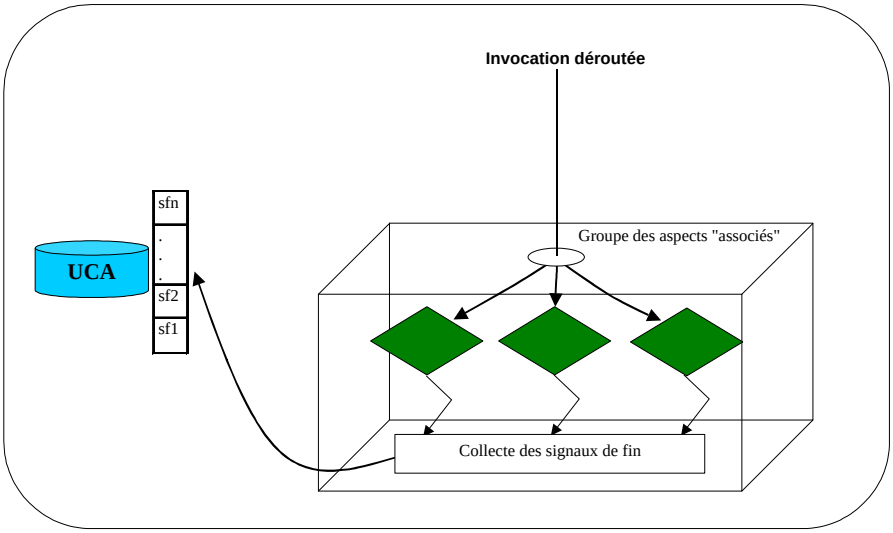

Figure V.12 Exécution des aspects "associés" en parallèle

Le routage de l'invocation à travers les différents composants de la fondation de contrôle se fait uniquement au travers de la méthode invoke(...), telle que :

`method.invoke(tool, args[])`

où :

- `method` est l'objet (au sens Java) représentant la méthode invoquée (instance de la classe `java.lang.reflect.Method`) ;
- `tool` est l'objet (au sens Java) représentant le composant sur lequel la méthode `method` doit être invoquée ;
- `args[]` est l'ensemble des arguments (avec leurs valeurs), paramètres de la méthode `method`.

Chaque aspect qui termine son exécution envoie un signal de fin au groupe des aspects "associés". Ce dernier, dès que le premier signal de fin lui est parvenu, déclenche l'invocation effective du service fédéral sur l'UCA (voir Figure V.12). L'exécution de chaque aspect va solliciter les rôles impliqués dans ces aspects.

```
>>> The ASPECT is invoking the role ...
=== The role WorkspaceManager is now invoked
=== ...and is identifiying which tool has to be invoked...
```

Exécution des rôles et identification des outils à solliciter

Le rôle *WorkspaceManager* est invoqué par un des aspects. Ce dernier détermine si le produit *p* transféré entre les deux activités est un document de spécification ou non.

```
public void transfert(Product p) {
        System.out.println("=== The role "+this.getName()+" is now invoked
                ...)\n");
        System.out.println("=== ...and is identifiying which tool has to be
                invoked...\n");
        if ( (p.getName()).startsWith("Spec")) {
              cvs.transfert(p);
              System.out.println("=== The role "+this.getName()+" invoked the
                    CVS proxy \n");
        }
        else {
              rcs.transfert(p);
              System.out.println("=== The role "+this.getName()+" invoked the
```

```
                    RCS proxy\n");
        }
}
```

Code du rôle *WorkspaceManager* permettant de déterminer l'outil à invoquer en fonction de la nature du produit

Sur l'exécution du code ci-dessus, le représentant de l'outil est invoqué.

```
=== The role WorkspaceManager is invoking the RCS proxy
=== The RCS proxy is performing all necessary translations product->file
=== The RCS proxy is invoking the RCS wrapper ...
```

Le nom du produit (retourné par la méthode `getName()` appelée sur l'objet *p*) est "doc_de_specs.doc". Il ne commence pas par la chaîne de caractère "spec". Suivant le code du rôle *WorkspaceManager*, c'est le représentant de l'outil RCS qui est sollicité.

Nous avons déjà détaillé les tâches du représentant : effectuer les conversions nécessaires permettant d'obtenir un *fichier* à partir d'un objet de type *produit*.

```
=== The RCS proxy is performing all necessary translations product->file
=== The RCS proxy is invoking the RCS wrapper ...
=== The RCS wrapper is calling the RCS tool for service request ...
before exec de c:\Program Files\ComponentSoftware\CS-RCS\System\csrcs CheckIn
c:\temp\doc_de_specs.doc ok
```

Le représentant appelle la façade de l'outil RCS à travers la méthode `checkIn(f)` où *f* est le fichier dont il faut effectuer la révision.

A l'exécution de cette méthode `checkIn(f)`, l'outil RCS est invoqué au travers de sa commande en ligne. L'exécution de cette commande provoque l'affichage d'une fenêtre pour information auprès de l'utilisateur, confirmant ainsi que le fichier (document de spécifications) a été ajouté à l'arborescence de RCS (cas qui se produit lorsque le fichier est révisé pour la première fois).

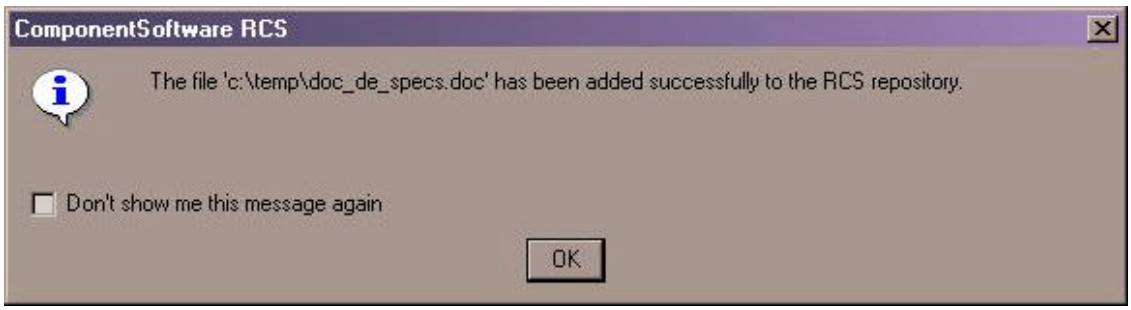

Figure V.13 Ajout d'un fichier dans la base des fichiers révisés par RCS

Lorsqu'on exécute le scénario une autre fois, sans modification de topologie au niveau des outils de la fédération, une fenêtre graphique différente apparaît, indiquant cette fois qu'il s'agit d'une autre révision et invitant l'utilisateur à saisir quelques informations (voir Figure V.13).

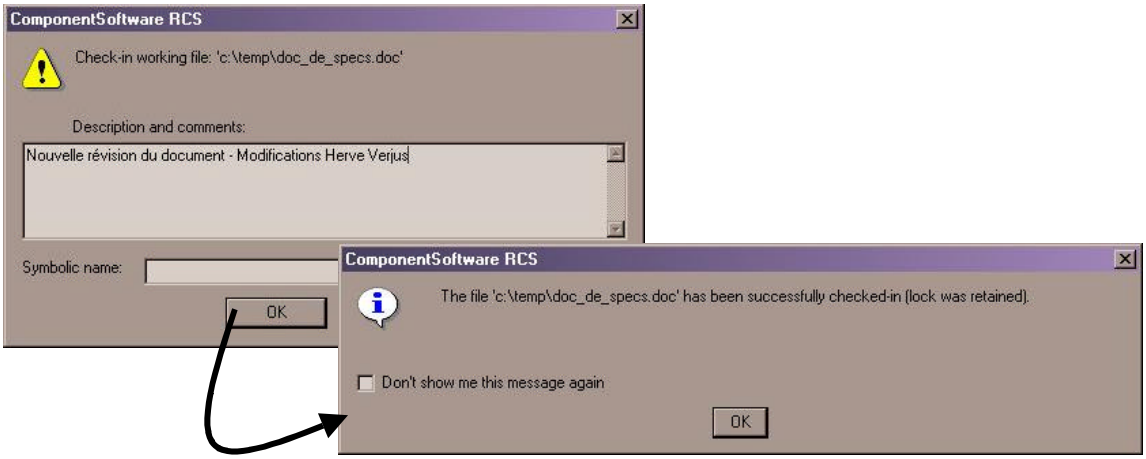

Figure V.14 Révisions ultérieures...

Nous notons que la façade de l'outil RCS exécutant les services RCS au travers de commandes en ligne a bien pour conséquence d'exécuter l'outil.

V.4.1.3 Exemple de traces générées à l'exécution

La figure V.15 montre l'exécution du scénario ainsi qu'un exemple de traces générées par ce dernier.

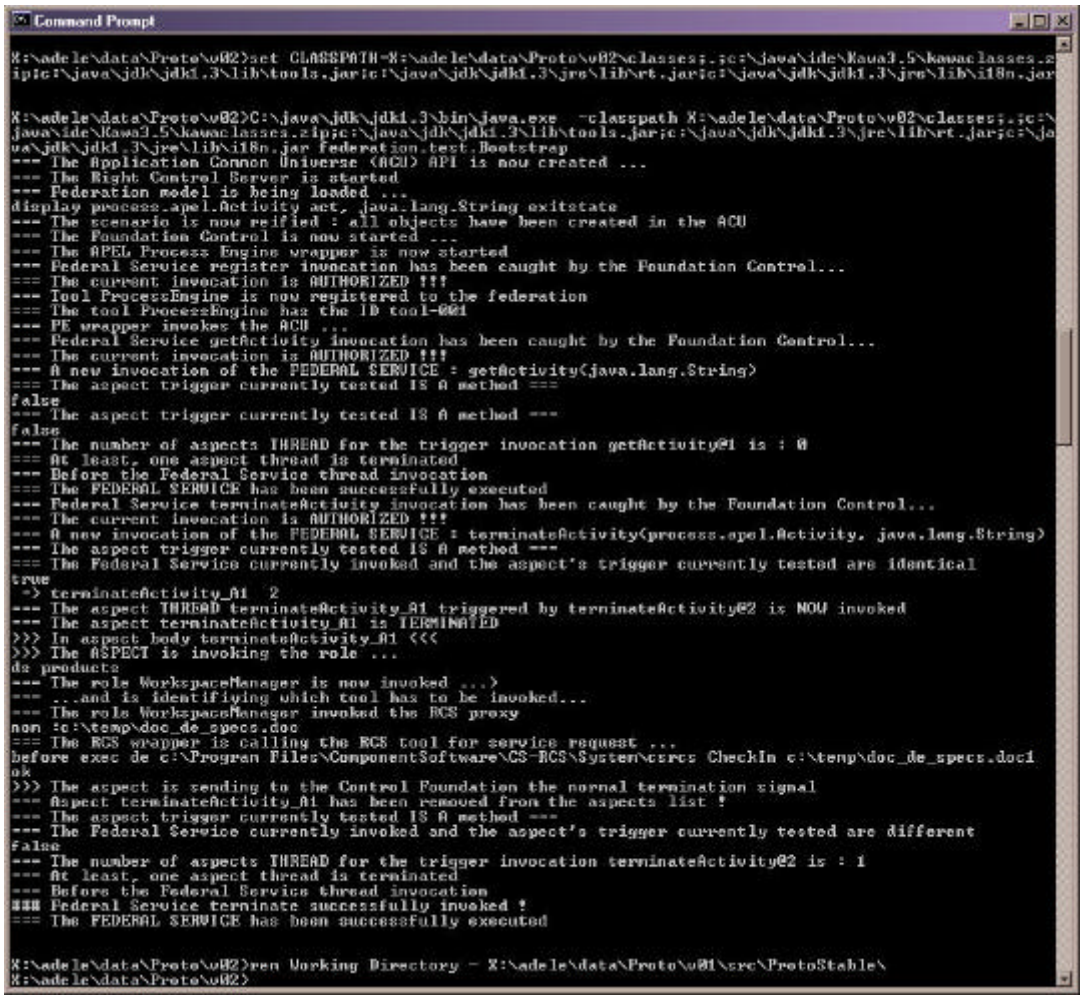

Figure V.15 Traces à l'exécution du scénario...

Le scénario a été exécuté avec des "politiques" de contrôle différentes. Les traces ainsi générées lors de ces exécutions sont regroupées dans la troisième partie de l'annexe B.

V.4.2 Validation

Les schémas de principe (voir annexe C) ont été en partie validés. Nous avons pu tester ce même scénario en appliquant plusieurs politiques de contrôle :

1. le cas de l'anarchie contrôlée (souscription des outils et contrôle des invocations uniquement),
2. le cas de la dictature où les invocations sont contrôlées et des aspects associés aux services fédéraux invoqués ont été déclenchés,
3. le cas bloquant où l'invocation est stoppée.

L'approche présentée au cours du quatrième chapitre a pu être validée (en tenant compte des limitations que nous avons citées) par l'exécution du prototype réalisé et le scénario mis en place. Nous verrons dans le chapitre portant sur les conclusions et les perspectives que depuis, des améliorations ont été apportées au prototype.

La version améliorée du prototype, pris en charge par Tuyet Lee Anh a été validée dans le cadre du projet ESPRIT IV LTR PIE (34840).

V.4.3 Distributions des outils de la fédération

L'une des limitations du prototype développé concernait la non prise en compte de la distribution des outils de la fédération. Cette dernière est désormais intégrée au prototype actuel, grâce aux technologies :

- Java/RMI pour la distribution des objets répartis et les appels/invocations synchrones ;
- une implémentation de JMS (Java Message Service) pour les communications asynchrones.

L'ensemble des prérogatives permettant le contrôle de la fédération et sur lesquelles s'appuyait la fondation de contrôle (les "proxy" Java) ont du être re-implémenté pour tenir compte de la distribution. En effet, les "proxy" Java ne fonctionnement pas avec les objets distribués.

Après que plusieurs solutions aient été envisagées, un double mécanisme de "proxy" Java a permis d'utiliser les "proxy" Java avec des objets répartis. L'ensemble des travaux qui ont permis l'implémentation de la solution retenue a été effectué par deux étudiants au cours d'un stage de fin de cursus (niveau ingénieur).

V.4.4 Mécanismes d'introspection

L'ensemble des vérifications pouvant être faites et portant sur le modèle de fédération :

- les services fédéraux de l'UCA ;
- l'implémentation des rôles ;
- l'endossement des rôles par les outils ;
- etc.

utilisent principalement les possibilités d'introspection fournies par le langage Java. Par exemple, le représentant *r* de l'instance *i* devra implémenter l'ensemble des interfaces des rôles endossés par cette instance : cette vérification se fait par introspection de la classe Java décrivant le représentant de l'instance en question.

L'interface de l'UCA est définie au travers d'une interface Java. Le concepteur de la fédération précise, lors de la phase initiale de la définition de la fédération, quelles sont les interfaces Java qui définissent les services fédéraux de l'UCA. Par introspection, l'ensemble des services fédéraux est alors automatiquement chargé pour utilisation ultérieure (association des aspects aux services fédéraux, etc.).

V.4.5 Les chiffres…

La première version du prototype, implémentant les mécanismes de base de la fondation de contrôle, permettant de gérer des modèles de fédérations décrits sous forme de classes Java et n'offrant aucun outil graphique à l'utilisateur (concepteur de fédérations d'outils), comportait 35 classes Java dont :

- 9 classes et interfaces Java concernant la définition du modèle de fédération ;
- 26 classes Java implémentant les mécanismes d'exécution et de contrôle de la fondation de contrôle.

Soit environ 3500 lignes de code Java. Le support des appels distants et des "proxy" Java amène le tout à 58 classes et plus de 5200 lignes de code Java.

Le scénario implémenté avec du code généré par l'éditeur d'APEL v5 comprend 15 classes Java.

V.4.6 Conclusion

Les premiers résultats apportés par la mise en place des scénarios nous paraissent intéressants :

- ils montrent la faisabilité de notre approche en s'appuyant sur les technologies Java ;
- ils montrent qu'il est possible de fournir dans un avenir proche, un certain nombre d'outils au concepteur l'aidant dans la définition de fédérations d'outils ;
- ils montrent que le prototype développé sert de support et facilite la construction de fédérations d'outils.

Aussi, d'autres efforts sont actuellement consentis pour poursuivre les travaux que nous avons entrepris.

Cependant, nous pensons que d'autres expériences sont nécessaires pour valider l'approche à des échelles plus importantes et en particulier, pour des cas d'utilisation industriels, impliquant des outils ou des systèmes plus complexes.

V.5 Vers un langage de définition de fédérations d'outils

La figure V.7 montre que les modèles de fédérations sont stockés sous forme de classes Java dans la version du prototype développée. Le langage Java est un langage de programmation

orienté objet n'offrant aucune extension particulière à la définition de fédération d'outils logiciels. Le concepteur de fédérations doit coder les modèles de fédérations d'outils en utilisant le langage Java, ce qui requiert une certaine expertise de la programmation.

Le but de cette section est de présenter le langage FTML (*Federation Tools Markup Language*) comme pouvant être un candidat à l'alternative du langage Java pour définir des modèles de fédérations d'outils. Le but de ce langage, dérivé du langage XML (*eXtended Markup Language*), est de fournir un langage dont les balises sont définies à partir des concepts présentés dans le quatrième chapitre et repris dans les paragraphes précédents (en particulier dans la section V.2). Pour le concepteur de la fédération, seule la structuration sous forme de document XML est à connaître ainsi que l'ensemble des balises et leur signification (c'est-à-dire la sémantique associée). Ceci évite d'avoir recours au langage Java qui n'a pas été conçu pour définir des fédérations d'outils, facilitant ainsi le rôle du concepteur.

Cependant, certains points ne sont que partiellement couverts par le langage : en particulier, la définition des aspects, des représentants et, dans un degré moindre, des rôles qui nécessitent un langage comprenant des structures de contrôles permettant de définir la coordination, les conversions, etc. Dans la version proposée du langage FTML, l'expression de la coordination, des conversions, etc. peut se faire en enveloppant du code Java dans certaines balises (appropriées) du langage. Ce langage permet :

- de bénéficier d'une syntaxe et d'une sémantique appropriées à la modélisation des fédérations d'outils basées sur les concepts proposés ;
- de faciliter la tâche du concepteur ;
- de permettre une grande lisibilité d'un modèle de fédération écrit en FTML (langage descriptif) sans avoir à "s'immerger" dans du code Java.

La grammaire du langage FTML est donnée dans l'annexe A sous forme de documents de DTD (Document Type Definition).

Nous allons présenter quelques éléments d'un modèle de fédération. Un modèle complet se trouve en annexe B.

V.5.1 Eléments pour définir un modèle de fédération

Pour définir un modèle de fédération d'outils, le concepteur doit utiliser les concepts décrits dans la section V.2. Le langage FTML utilisant les mêmes concepts, le modèle global d'une fédération d'outils va comporter les éléments suivants :

- la définition des services fédéraux de l'UCA, définis à l'intérieur de l'élément CU-SET ;
- la définition des rôles à l'intérieur de l'élément ROLES ;
- la définition des outils comprenant :
  - la définition des modèles d'outils à l'intérieur de l'élément MODELS ;
  - la définition des instances à l'intérieur de l'élément INSTANCES ;
  - l'attribution des rôles aux instances à l'intérieur de l'élément ROLE-COOPERATION ;
- la définition des représentants à l'intérieur de l'élément PROXIES ;
- la définition des façades à l'intérieur de l'élément WRAPPERS.

Nous notons que les façades simples peuvent être partiellement obtenues par déduction des modèles d'outils (voir section V.2.6).

Ainsi, un modèle de fédération d'outils est défini selon le document de DTD suivant :

```
<!ELEMENT FEDERATION (
                       CU-SET,
                       ROLES?,
                       ASPECTS?,
                       SITES,
                       MODELS,
                       INSTANCES,
                       ROLE-COOPERATION?,
                       PROXIES?,
                       WRAPPERS?
                       )>

<!ATTLIST FEDERATION beginDate  CDATA                          #IMPLIED>
<!ATTLIST FEDERATION endDate    CDATA                          #IMPLIED>

<!ATTLIST FEDERATION priority   (prohibition | right) "right"   #IMPLIED>

<!ENTITY % CU-SET               SYSTEM          "cu-set.dtd">
<!ENTITY % ROLES                SYSTEM          "roles.dtd">
<!ENTITY % ASPECTS              SYSTEM          "aspects.dtd">
<!ENTITY % SITES                SYSTEM          "sites.dtd">
<!ENTITY % MODELS               SYSTEM          "models.dtd">
<!ENTITY % INSTANCES            SYSTEM          "instances.dtd">
<!ENTITY % ROLE-COOPERATION     SYSTEM          "role_cooperation.dtd">
<!ENTITY % PROXIES              SYSTEM          "proxies.dtd">
<!ENTITY % WRAPPERS             SYSTEM          "wrappers.dtd">
```

Eléments du langage FTML permettant la définition d'un modèle pour une fédération d'outils

Chacun de ces éléments est défini par un document de DTD spécifique se trouvant dans l'annexe A. Ces documents précisent la syntaxe complète des balises et les caractéristiques prises en compte pour la modélisation des fédérations d'outils.

Nous allons, dans le paragraphe suivant, prendre l'exemple de l'outil RCS et proposer son modèle à l'aide du langage FTML.

V.5.2 Le modèle d'outil

L'ensemble des caractéristiques prises en compte pour la définition du modèle de l'outil RCS (voir la description du modèle page suivante) sont exprimées à l'intérieur des balises <MODEL> ... </MODEL>. Ces balises sont elles-mêmes des balises incluses dans l'élément "père" MODELS de la définition du modèle global (ce modèle regroupe l'ensemble des modèles d'outil) d'une fédération d'outils.

```
<?xml version="1.0"?>
<!DOCTYPE MODEL SYSTEM "model.dtd">


<MODEL modelID="RCSforWindows" client-server="no" portable="no"
   persistent-state="yes" modifyFS="yes" generates-events="no"
   user-interaction="yes">
   <MODEL-DESC>A well known version manager</MODEL-DESC>
   <WnP-SERVICE-INTERFACE>
      <COMMAND-INTERFACE>
         <COMMANDS>
            <COMMAND commandID="createRCSFile"
                comment="create a new file" transactional="no">
```

```
                <NAME>CSRCS create</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
            <COMMAND commandID="checkIn"
                comment="check-in a modified file" transactional="no">
                <NAME>CSRCS CheckIn</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
            <COMMAND commandID="checkOut"
                comment="check-out a working copy" transactional="no">
                <NAME>CSRCS CheckOut</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
            <COMMAND commandID="history"
                comment="trace the revisions file history" transactional="no">
                <NAME>CSRCS History</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
         </COMMANDS>
         <InitC-COMMANDS>
            <COMMAND-ID>createRCSFile</COMMAND-ID>
         </InitC-COMMANDS>
      </COMMAND-INTERFACE>
   </WnP-SERVICE-INTERFACE>
   <CONTEXT contextID="win">
      <PLATFORM processorArch="i486" os="Windows" version="NT 4"></PLATFORM>
      <BINARY path="c:\Program Files\CSRCS\">CSRCS</BINARY>
   </CONTEXT>
</MODEL>
```

Description du modèle de l'outil RCS

Par exemple, nous retrouvons la description du service "checkIn" (voir partie de la figure mise en gras). Cette description montre que :

- ce service est une "commande en ligne" ;
- la syntaxe d'invocation est "CSRCS CheckIn" ;
- la commande admet un paramètre obligatoire qui est un fichier (le nom du fichier à réviser) ;
- le service n'est pas un service pouvant être exécuté en mode transactionnel[69].

Notons que la description ci-dessus tient compte de la taxonomie des services :

- commandes en ligne (balises <COMMAND> ... </COMMAND>) ;
- messages (balises <MESSAGE> ... </MESSAGE>) ;
- opérations (balises <OPERATION> ... </OPERATION>).

Dans la description des modèles d'outils, certains des services identifiés pourraient appartenir à une interface logicielle au travers de laquelle les services peuvent être directement invoqués à distance (exemple d'une interface distante Java). Au contraire, d'autres services doivent nécessairement être enveloppés pour qu'ils puissent être invoqués à distance. Dans ce cas, c'est le rôle des façades et le concepteur s'appuie sur le modèle de l'outil pour identifier les services qui doivent être enveloppés et ceux qui n'ont pas besoin de l'être.

[69] C'est-à-dire que l'outil RCS ne permet pas de retour arrière après l'invocation de ce service.

V.5.3 Illustration des "paradigmes" et influence d'un changement

Un rôle, dont la description est délimitée par les balises <ROLE> ... </ROLE> (voir description du rôle *WorkspaceManager*), est défini au travers de devoirs et d'interdictions. Il s'agit donc d'un modèle comportemental que devront adopter tous les outils jouant le rôle en question. L'ensemble des rôles est compris à l'intérieur de l'élément ROLES (voir section V.5.1) du modèle global d'une fédération d'outils.

```
<?xml version="1.0"?>
<!DOCTYPE ROLE SYSTEM "role.dtd">


<ROLE roleID="WorkspaceManager">
    <DUTIES>
        <ROLE-SERVICES>
            <OPERATION operationID="productTransfert"
      comment="a product transfert between source activity and a target one"
      transactional="no" returnType="void">
                <NAME>transfert</NAME>
                <PARAMETER type="Product" name="p">
            </OPERATION>
            <OPERATION operationID="fileTransfert"
      comment="a transfert operation between source activity and a target one"
      transactional="no" returnType="void">
                <NAME>transfert</NAME>
                <PARAMETER type="String" name="f">
            </OPERATION>
        </ROLE-SERVICES>
    </DUTIES>
    <PROHIBITIONS>
        <FEDERAL-SERVICE-CALLS>
            <FEDERAL-SERVICE-ID>terminateActivity</FEDERAL-SERVICE-ID>
        </FEDERAL-SERVICE-CALLS>
    </PROHIBITIONS>
</ROLE>
```

Description du rôle "gestionnaire d'espaces de travail"

Dans l'exemple, nous avons défini un rôle "*WorkspaceManager*" dans lequel lui sont attribués des devoirs et des interdictions. Par exemple, le rôle fournit un service appelé "productTransfert" dont la syntaxe est "`transfert(Product p)`". Cet exemple couvre le mode de fonctionnement "dictatorial" présenté dans la section IV.3.4. Ce mode exigeait la définition d'un aspect impliquant le rôle "*WorkspaceManager"*. La définition de l'aspect est donnée dans la section IV.3.6. Quant aux interdictions (balises <PROHIBITIONS>...</PROHIBITIONS>), l'exemple montre qu'une interdiction a été définie sur l'utilisation du service fédéral "terminateActivity".

La définition du rôle "*WorkspaceManager*" est telle que pour chaque outil (instance) endossant ce rôle cela signifie :

- qu'il est impliqué dans le service fédéral "terminateActivity" (au travers de l'aspect associé à ce service fédéral),
- qu'il ne pourra pas invoquer le service fédéral "terminateActivity".

Passer de la dictature à l'anarchie ?

Une alternative parmi d'autres possibilités consisterait à ne pas définir d'aspect pour ce service fédéral et à faire souscrire les outils à l'utilisation du service fédéral[70]. Selon cette alternative, les outils bénéficieraient de plus de "libertés" : il suffit de supprimer la description de l'aspect et de modifier la description du rôle *WorkspaceManager* en précisant des souscriptions et non des services à fournir, comme le montre la description suivante.

```
<?xml version="1.0"?>
<!DOCTYPE ROLE SYSTEM "role.dtd">


<ROLE roleID="WorkspaceManager">
    <DUTIES>
        <ROLE-SUBSCRIPTIONS>
            <FEDERAL-SERVICE-ID>terminateActivity</FEDERAL-SERVICE-ID>
        </ROLE-SUBSCRIPTIONS>
    </DUTIES>
    <PROHIBITIONS>
        <FEDERAL-SERVICE-CALLS>
            <FEDERAL-SERVICE-ID>terminateActivity</FEDERAL-SERVICE-ID>
        </FEDERAL-SERVICE-CALLS>
    </PROHIBITIONS>
</ROLE>
```

Modification du comportement du rôle "*WorkspaceManager*"

Nous remarquons en outre que la définition d'un rôle permettant d'implanter une politique "anarchiste" est relativement simple (minimaliste).

V.5.4 Conclusion

La définition du langage FTML nous a permis de procéder à certaines vérifications des caractéristiques pertinents à prendre en compte dans le modèle de fédération d'outils. En effet, la description du modèle d'outil RCS permet à sa simple lecture, de constater l'adéquation avec l'outil "réel" ou, au contraire l'inadéquation des caractéristiques modélisées.

Le langage FTML prend en compte de nombreuses caractéristiques qui ne sont pas exploitées dans le prototype utilisant des classes Java. L'utilisation du langage n'a pas été effective. La raison principale est qu'à l'état actuel des travaux et comme nous l'avons déjà signalé, d'autres expérimentations doivent être menées avant d'entreprendre des réalisations concrètes que ce soient des outils, des générateurs ou des langages. Une autre raison est que la définition d'une fédération d'outils basée sur l'utilisation d'un langage spécifique n'est qu'une alternative au modèle décrit sous forme de classes Java. D'autres options existent parmi lesquelles nous pouvons citer des formalismes graphiques ou encore des formalismes plus formels. L'état actuel de nos travaux et les expérimentations qui sont envisagées ne nous permettent pas de nous avancer sur les formalismes qui seront réellement utilisés pour décrire des fédérations d'outils et dans lesquels seront matérialisés nos concepts.

[70] Ce qui signifie que dès que le service fédéral sera invoqué, les outils endossant le rôle "souscripteur" devront recevoir une notification.

Chapitre VI CONCLUSIONS ET PERSPECTIVES

Dans cette thèse, nous avons traité le problème des fédérations d'outils logiciels. L'objectif de ces travaux est de fournir une approche permettant de concevoir des fédérations d'outils et de proposer un cadre pour leurs mises en oeuvre. A travers l'approche proposée, plusieurs problèmes qui n'étaient pas traités simultanément, ont pu être pris en compte et gérés selon une même approche.

Notre proposition et l'implémentation réalisée au travers d'un prototype permettent :

- d'établir une description d'une fédération d'outils ;
- de mettre en place une architecture supportant la fédération d'outils et dont le fonctionnement est conforme à la description fournie.

Il est alors possible, pour l'utilisateur, de construire un système cohérent sur la base d'outils hétérogènes, autonomes, souvent issus du marché, qui vont être distribués sur le réseau. La problématique est relativement complexe (particulièrement l'intégration d'outils de type COTS [Carney 1997], DeLine 1999, Balkt et Kedia 2000]). Les expériences et travaux menés jusqu'alors pour atteindre ces objectifs soient ne prennent pas simultanément en compte l'ensemble des contraintes (hétérogénéité, distribution, autonomie, etc.), soient abordent le sujet selon une approche bas niveau ; cela constitue un frein à l'émergence de solution adaptée [Estublier et al. 2001a], attendue des utilisateurs et des entreprises, répondant à une problématique concrète.

Nos travaux couvrent essentiellement trois facettes :

- la description des outils. Nous avons recensé un ensemble de caractéristiques à prendre en compte ;
- la mise en place d'un mécanisme de contrôle ;
- une couche d'adaptation permettant aux outils d'intégrer une fédération selon une approche "boîte noire".

VI.1 Bilan et remarques sur l'approche

Nous avons, dans le quatrième chapitre, présenté l'architecture de base d'une fédération et sa topologie. Ainsi, une fédération est définie par l'existence d'un univers commun et d'un ensemble d'outils nécessaires à l'application, regroupés au sein de ce que nous avons appelé une fondation de l'application [Estublier et Verjus 2001, Estublier et al. 2001a]. Nos objectifs couvrent à la fois la proposition d'une architecture flexible et modulable, mais également un ensemble de concepts permettant de définir les fédérations d'outils.

Concernant la coopération, elle n'est gérée (et définie) qu'au travers de la fondation de contrôle qui détermine quels sont les outils impliqués, dans quels aspects associés à quels services fédéraux. D'une part, les outils ne se connaissent pas (les outils de type COTS n'ont pas de dépendance entre eux a priori) et d'autre part, ce n'est pas à l'autorité commune (la fondation de contrôle) de gérer explicitement les interactions directes entre outils (l'hypothèse faite dans le cadre des fédérations préserve justement l'autonomie). Cela signifie que s'il existe des interactions directes entre outils, elles ne sont, en aucun cas, prises en compte dans le cadre de nos travaux.

VI.1.1 La conception de fédérations d'outils

Le concepteur de fédérations d'outils dispose d'un ensemble de concepts. L'utilisation de ces concepts permet de modéliser des fédérations d'outils selon :

1. les concepts communs de l'application à réaliser ;
2. les outils logiciels participants ;
3. le fonctionnement de la fédération, le type de contrôle et la cohérence pour que les outils participants réalisent l'application.

Nous retrouvons les axes décrits dans [Bolusset et al. 1999], c'est-à-dire, décrire le *quoi*, le *qui* et le *comment*.

Le premier point est couvert par le concept d'*univers commun* et les *services fédéraux* associés à cet univers commun.

Le second point est couvert par la description des *modèles d'outils* utilisés ainsi que les *instances* basées sur ces modèles.

Le troisième point comprend deux axes : (1) le contrôle de la fédération en terme de paradigme de fonctionnement, de comportement fonctionnel, pris en charge par la description des *rôles*, des *aspects* et (2) l'adaptation des outils, prise en charge par la description des *représentants* et des *façades*.

Les paragraphes précédents ont présenté les concepts et leur utilisation, par le concepteur de fédérations, sous un angle pratique (implémentations réalisées dans le cadre du prototype que nous avons développé).

Concepts	Caractéristiques	Inconvénients ou difficultés
Univers Commun	– "espace" de synchronisation – connaissances (concepts) communes – abstraction de concepts concrets	– difficile à définir
Rôle	– vision abstraite des comportements, (relativement) indépendante de l'implémentation – simple à définir	– recenser les services "pertinents" – allers-retours avec les outils "réels"
Modèle d'outil	– modélisation des outils – taxonomie des services fournis – niveau de description : o intrinsèque – simple à définir si documentation des outils complète	– choix des caractéristiques à modéliser – exploitation de ces caractéristiques dans le modèle de fédération
Aspect	– coordination des différents **rôles** (vision abstraite) – réglage du contrôle (transactions, contraintes, etc.)	– nécessite un langage permettant d'exprimer la coordination – choix sur la "dureté" du contrôle (propriétés)
Instance d'outils	– "hérite" d'un modèle d'outil – comprend l'attribution des rôles – niveau de description : o contextuel	
Représentant	– expression des conversions entre les concepts de l'UCA et les concepts propres à chaque outil – bibliothèques d'opérations de conversion	– nécessite un langage permettant d'effectuer les conversions – expertise requise pour définir les conversions
Façade	– indépendantes de fédérations particulières (si outils associés à des représentants) – se basent sur des mécanismes génériques, communs à l'ensemble des outils – leur production peut être partiellement automatisée	– peuvent perdre leur caractère générique si l'outil ne dispose que d'une façade et pas de représentant

Tableau VI.1 Récapitulatif des concepts

A partir du tableau VI.1, nous identifions les difficultés particulières à la mise en oeuvre de fédérations d'outils concernant :

- la définition de l'UCA ;
- la définition des aspects ;
- la définition des représentants.

Elles font appel à un niveau d'expertise relativement élevé de la part du concepteur ou/et sont intrinsèquement complexes (apprentissage de langages pour exprimer la coordination, les conversions, etc.). Par exemple, certaines tâches semblent, pour l'instant, ne pas pouvoir être automatisées (les opérations de conversions [Busse et al. 1999], etc.).

Par contre, d'autres concepts sont relativement faciles à utiliser :

- la définition des rôles ;
- la définition des outils (modèles d'outils et instances) ;
- la définition des façades.

L'approche adoptée montre que certaines facettes de la modélisation des fédérations d'outils sont relativement simples à définir alors que d'autres restent un peu plus complexes. Notre approche à permis de dégager deux niveaux de description des outils :

- la **description intrinsèque** qui ne tient aucun compte du contexte dans lequel va être exécuté l'outil ;

- la **description contextuelle** qui définit le comportement d'un outil spécifique dans une fédération spécifique.

Cette dichotomie permet d'avoir un pouvoir expressif plus important, de séparer logiquement les informations liées au contexte de la fédération de celles permettant de définir un outil appartenant à un environnement quelconque et, surtout, cela permet de pouvoir réutiliser une grande partie de la modélisation des outils dans d'autres contextes, d'autres fédérations. En effet, les modèles d'outils (description intrinsèque) sont génériques :

- ils sont indépendants d'une fédération spécifique (ré-utilisation du même modèle d'outil d'une fédération à une autre) ;
- ils sont indépendants du domaine d'application portant sur les fédérations (ré-utilisation potentielle dans d'autres domaines que celui des fédérations de progiciels).

VI.1.2 Une architecture pour la mise en oeuvre de fédération d'outils

L'architecture que nous proposons comme support aux fédérations d'outils logiciels repose sur trois fondations :

- la **fondation de l'application** contient les bases d'une fédération d'outils (le quoi et le qui). Par contre, une fédération qui se résumerait à cette fondation ne contient aucun modèle de fédération, aucune description ne serait-ce que des outils participants. D'autre part, cette fédération répond à un style "anarchique" puisque aucun contrôle n'est défini explicitement ;
- la **fondation de contrôle** qui dispose d'un univers commun de contrôle contenant le modèle de fédération (c'est-à-dire, la description des outils de la fédération ainsi que le modèle de contrôle - explicite, c'est-à-dire, comment doit fonctionner la fédération à l'exécution). Cette fondation comprend également un ensemble de composants qui interprètent le modèle de fédération et qui appliquent les contraintes liées au contrôle lors de l'exécution ;
- la **fondation du processus** dont l'univers commun contient le processus d'évolution de l'univers commun de la fondation de l'application. La fondation du processus régule en quelque sorte, le comportement de la fondation de l'application par la description et la mise en oeuvre de l'évolution attendue de l'état de l'UCA.

Aussi, l'architecture que nous proposons peut être à géométrie variable ; elle peut ou non comporter une fondation de contrôle, une fondation du processus. Une architecture comprenant plusieurs fondations de même nature possède les caractéristiques suivantes :

- l'univers commun de l'application sera distribué sur l'ensemble des fondations de l'application ;
- le contrôle de la fédération sera distribué sur l'ensemble des fondations de contrôle ;
- le contrôle de la fédération par le processus sera distribué sur l'ensemble des fondations du processus.

VI.1.3 Une vision selon deux axes

En effet, nous considérons deux axes (voir section IV.7.1) pour construire et mettre en oeuvre des fédérations. Le premier (horizontal), selon lequel une fédération peut être vue comme un ensemble de

trois fondations de natures différentes. Le second (vertical), pour lequel nous proposons un ensemble de concepts permettant de passer d'une vision abstraite de la fédération (UCA et rôles) à son implémentation (instances d'outils et leurs représentants). Nous pensons que cette vision selon ces deux axes est une des originalité des travaux menés.

VI.1.4 Un langage de description des fédérations d'outils

Dans le cadre des travaux de cette thèse, nous avons défini le langage FTML (*Federation Tools Marckup Language*), dérivé du langage XML, permettant de décrire des fédérations d'outils logiciels. Entre autres, les concepts de l'approche sont les principaux concepts du langages FTML. Ce langage permet de satisfaire deux objectifs :

1. recenser les concepts et les informations pertinents pour la modélisation des fédérations d'outils ;
2. offrir au concepteur de fédérations d'outils, un langage adapté pour décrire des fédérations d'outils et permettant une lecture simple et intuitive.

Le premier point a constitué un premier niveau de validation de notre approche dans le sens où la description d'une fédération d'outil définie avec les concepts du langage fournit un résultat cohérent.

Le second point a permis d'avoir une alternative à la description d'un modèle de fédération disséminée dans des classes Java et donc, difficilement lisible.

Cependant, ce langage n'est utilisé que par son apport descriptif. Il n'apporte rien au prototype et aux expérimentations que nous avons menées. En effet, aucun compilateur n'a été réalisé permettant, à partir d'une description FTML d'une fédération d'outils, de générer les fichiers classes Java correspondant.

VI.1.5 Validation de l'approche

L'approche que nous proposons a été validée à travers des expérimentations basées sur un scénario. D'autre part, le prototype actuel, basé sur la version décrite dans ce document et sur l'approche que nous avons présentée a été démontré dans le cadre du projet ESPRIT PIE. Des travaux de recherche, en parallèle, ont été menés sur les approches et les technologies permettant de distribuer l'ensemble des composants d'une fédérations (les outils participants et les composants de contrôle) [Cugola et al. 1999, Becchi et al. 2000, Cugola et al. 2000a, Cugola et al. 2000b, Cugola et al. 2000c].

Ces validations concernent essentiellement le scénario de transfert de produit entre deux activités d'un processus logiciel. Ce processus est l'application de la fédération qui comporte quelques outils dont certains de type COTS (RCS, CVS, MS-Word, etc.).

VI.2 Bilan et remarques sur les fédérations d'outils et l'état de l'art

VI.2.1 Développement de systèmes à base de COTS

Le tableau VI.2 reprend les critères énoncés dans l'état de l'art et résume le positionnement de notre proposition par rapport à ces critères.

Critères / Domaines	CBSE	EAI	SMA	Interopérabilité	SIF	Fédérations d'outils
modélisation des composants	oui	non	faible	non	faible	concepts de modèle d'outils et instance d'outils
autonomie des composants	non	oui	non	non	faible	oui
approche intrusive ou non intrusive	intrusive	non intrusive	intrusive	intrusive et non-intrusive	intrusive et non intrusive	non intrusive (pas de modification de code source des outils)
approche déductive ou inductive	déductive	inductive	déductive	inductive	déductive	mixte
interopérabilité syntaxique			oui	oui	oui	prise en charge au niveau des représentants
interopérabilité sémantique			faible	oui	oui	prise en charge au niveau des représentants
flexibilité du contrôle	non	non	non	non	non	concepts de rôle et d'aspect
paradigmes d'assemblage	non	utilise	oui	oui	non	combinaison des représentants, des façades et des aspects permettant de mettre en oeuvre différents paradigmes d'assemblages
moyens techniques	oui	oui	peu	non	oui	implémentation de la fondation de contrôle
modélisation globale	arch. logicielle	non	non	non	non	l'ensemble des concepts – langage FTML

Tableau VI.2 Les fédérations d'outils logiciels par rapport à l'état de l'art

L'approche permet de modéliser et de tenir compte de la modélisation des participants de la fédération : les outils selon un point de vue intrinsèque et un point de vue contextuel.

La participation des outils n'entrave pas leur autonomie (technique : aucune modification de leur code source, organisationnel : préservation du contexte et de l'environnement d'origine). Leur adaptation se fait en ayant recours à des composants intermédiaires (les représentants et les façades). Ces composants intermédiaires peuvent également être réutilisés par ailleurs (dans d'autres fédérations) et assurent l'interopérabilité entre chaque outil et l'univers commun (de l'application).

La fondation de contrôle (interprétant le modèle de fédération) est le moyen technique qui permet de mettre en oeuvre le contrôle des fédérations d'outils. Toute fondation de contrôle s'intercale entre les outils et l'univers commun de l'application de manière à intercepter les invocations dans les deux sens.

Notre approche, pour des fédérations d'outils, est la seule à prendre en compte l'ensemble des critères présentés dans le tableau VI.2. A la fois, il s'agit de la seule approche permettant de modéliser un système complet (le quoi, le qui, le comment) avec un niveau d'abstraction relativement élevé, et de mettre à la disposition des utilisateurs des moyens (fondation de contrôle, fondation du processus) qui implémentent la modélisation. Si nos objectifs sont proches de ceux des approches dans le domaine de l'EAI, notre démarche diffère cependant des solutions apportées dans ce domaine qui sont de bas niveau, privilégiant les moyens techniques à la conception et à la construction de systèmes intégrant des applications hétérogènes.

En ce qui concerne les systèmes à base de composants, les travaux dans ce domaine tentent de proposer des spécifications à partir desquelles les composants sont développés (adhésion à un modèle spécifique). Cette démarche vient à l'encontre de celle que nous avons adoptée pour laquelle, au contraire, aucune hypothèse n'est faite sur l'origine des applications, ni même sur les spécifications à partir desquelles elles ont été développées.

Enfin, les travaux dans le domaine des systèmes à base de composants de type COTS sont majoritairement orientés vers la phase de l'analyse des besoins, la caractérisation du processus

de développement spécifique à de tels systèmes, la qualification des composants, leur sélection et les résultats (quantitatifs et qualitatifs) déduits de leurs utilisations. Par contre, peu d'efforts sont entrepris pour fournir des moyens techniques et conceptuels permettant à des outils COTS d'intégrer un système plus large. Comme nous l'avons écrit dans le quatrième chapitre, notre démarche s'inscrit dans cette perspective en espérant que d'autres travaux viendront peu à peu combler le fossé que nous avons constaté [Estublier et al. 2001a, Estublier et al. 2001b].

VI.2.2 Les réponses aux problèmes recensés

Nous avons recensé quatre problèmes inhérents aux systèmes intégrant des applications hétérogènes dont font partie les fédérations d'outils [Estublier et al. 2001a] :

- problème de recouvrement ;
- problème d'indéterminisme ;
- problème d'incohérence ;
- différents niveaux d'abstraction.

Notre proposition permet d'aborder ces quatre problèmes et fournit un certain nombre de moyens permettant de les gérer (voir quatrième chapitre). Ainsi, pour le problème de recouvrement, l'univers commun est le lieu où les concepts communs sont exprimés, selon un haut niveau d'abstraction. Seules les informations strictement nécessaires au fonctionnement de la fédération sont exprimées dans l'univers commun permettant aux différents outils d'en tirer leur propre "vision".

Les moyens permettant le contrôle des fédérations (basés sur les concepts de *rôle* et d'*aspect*) permettent de gérer l'indéterminisme lié aux comportement des outils et de leurs utilisateurs (contrôle individuel de chaque outil) et l'indéterminisme lié au comportement global de la fédération (contrôle collectif).

Enfin, la gestion de l'incohérence et la gestion des différents niveaux d'abstraction se fait au niveau des représentants essentiellement qui prennent en charge les conversions syntaxiques, sémantiques et qui font le lien entre les concepts de l'univers commun (concepts de haut niveau) et leurs implémentations au niveau des outils.

VI.2.3 Le mélange des modèles d'intégration

Plusieurs modèles ont été recensés permettant l'intégration d'applications logicielles ou de composants [Hayden et al. 1999, Mularz 1995, Stonebraker 1999] que nous avons présentés dans le troisième chapitre (section III.4.2). Ces modèles sont généralement utilisés exclusivement les uns des autres, suivant le paradigme d'interaction que le concepteur veut mettre en oeuvre ou/et le degré de contrôle. Il en allait de même pour les architectures proposées dans [Estublier et al. 1998b].

Les systèmes intégrant différents outils reposent sur une architecture, elle-même mettant en oeuvre un seul paradigme d'interaction. L'architecture que nous proposons, sur laquelle reposent les fédérations d'outils, au contraire, met en oeuvre plusieurs modèles d'intégration et, entre autre, permet d'avoir si le concepteur le souhaite, une dichotomie entre :

- l'encapsulation (de la partie fonctionnelle) de l'outil, basée sur les caractéristiques propres de l'outil assurée par la façade ;
- les conversions syntaxiques et sémantiques effectuées par le représentant.

En comparaison avec les modèles proposés par [Hayden et al. 1999], nous retenons que :

- les façades que nous proposons sont très proches des façades proposées par [Hayden et al. 1999] ;
- les représentants que nous proposons ont des points communs avec les ambassadeurs décrits dans [Hayden et al. 1999].

L'originalité, dans le cadre de nos travaux, provient de l'association des deux (représentants et façades) permettant au concepteur d'avoir un plus grand contrôle (combinaison des paradigmes) et une plus grande lisibilité sur les tâches incombant aux façades ou aux représentants (problèmes évoqués par [DeLine 1999] avec le mélange de la partie fonctionnelle et celle portant sur les interactions).

VI.3 Bilan et remarques sur les expérimentations

Afin de valider notre approche, un prototype a été développé. Un scénario (avec plusieurs variantes) a été exécuté pour montrer la concrétisation pratique de notre approche davantage théorique. Les expérimentations ont été cependant limitées et en quantité et en complétude.

Le comportement du prototype, lors de l'exécution du scénario, a été conforme à ce que nous en attendions. En revanche, nous pensons que d'autres expérimentations doivent être menées afin de tirer davantage de conclusions et d'entrevoir des perspectives plus ciblées. En particulier, les futures expérimentations doivent prendre en compte :

- des outils de type COTS de taille plus importante ;
- une application réelle issue, par exemple, d'une problématique industrielle ;
- une réelle distribution des outils.

Cela nous permettra de mesurer avec précision les efforts que doit consentir un utilisateur pour concevoir et construire une fédération d'outils selon notre démarche et de comparer ces efforts avec ceux qui sont nécessaires, selon une approche plus "triviale" ou de bas niveau. Pour résumer, cela nous permettra de répondre avec davantage de certitude si notre approche permet des gains :

- en termes de moyens,
- de compétences et d'expertises,
- de temps.

Au vue des expérimentations déjà effectuées, nous pensons que notre proposition simplifie de façon significative, l'intégration d'outils hétérogènes, se distinguant ainsi des propositions du type CCM par exemple qui tendent à devenir de plus en plus complexes.

En ce qui concerne la complétude, le prototype développé et utilisé pour les premières expérimentations comportait un certains nombre de limitations (voir section V.4.1.1). Ainsi, les aspects liés à la distribution ont été rapidement pris en compte. Le développement actuel est pris en charge par Tuyet Lee Anh dans le cadre de son travail de thèse, qui poursuit les travaux que nous avons entamés ; sa contribution au développement du prototype est importante (notamment par le développement d'un certain nombre d'outils graphiques ainsi que l'ajout d'un univers commun de contrôle persistant). Un prototype a été également réalisé permettant de tracer l'ensemble des actions de la fondation de contrôle.

Au cours du développement du prototype, nous avons été confrontés à certaines difficultés techniques. Comme nous l'avons vu, l'un des mécanismes importants de la fondation de

contrôle est d'intercepter les invocations en direction de l'univers commun et effectuées par les outils de la fédération (et seulement ceux la) ; puis, d'appliquer une politique de contrôle suivant l'appelant. Dans le cadre de la programmation "classique", l'invocation d'une opération (service synchrone) inhibe la possibilité d'identifier l'appelant à partir de l'appelé. Nous avons contourné cette limitation avec l'utilisation des "proxy" java.

Enfin, l'implémentation du prototype respecte l'approche adoptée, notamment en ce qui concerne sa topologie et la place de la fondation de contrôle : les composants du prototype de la fondation de contrôle se répartissent de part et d'autre de la couche réseau (couche de communication). Comme nous l'avons vu, nous avons d'une part les façades des outils (chaque façade et l'outil qui lui est associé s'exécutent sur la même machine) et d'autre part les autres composants que sont l'ensemble des serveurs (d'outils, de vérifications des droits, etc.) qui eux, s'intercalent entre la couche réseau et le composant "univers commun" (le SGBDOO). Ceci permet donc aux concepteurs de fédérations d'outils de bien placer les classes java.

VI.4 Perspectives

Les perspectives ouvertes par nos travaux peuvent se grouper selon trois axes (cf. Figure VI.1) : la continuation du travail proprement dit, des nouvelles applications et des nouveaux thèmes de recherche ouverts.

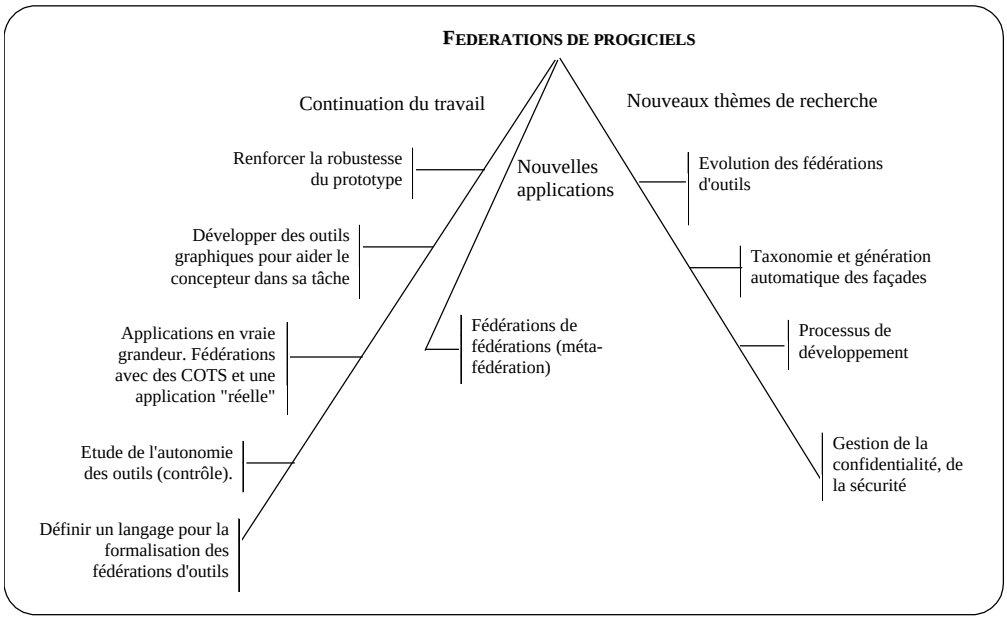

Figure VI.1 Perspectives de notre travail

Nous allons par la suite développer les perspectives mentionnées selon les trois axes.

VI.4.1 Continuation du travail entrepris

Développement du prototype

Le premier pas dans la continuation du travail concerne le développement de l'environnement permettant à l'utilisateur définir des fédération de progiciels, afin d'augmenter la facilité d'utilisation et sa robustesse. Le prototype présenté dans ce document de thèse a fait l'objet de développement constant depuis plusieurs mois au laboratoire IMAG-LSR et la version

actuelle présente un certain nombre de fonctionnalités qui n'ont pas été présentées dans ce document.

Entre autre, un ensemble d'interfaces graphiques permettent à l'utilisateur de définir une fédération d'outils à l'aide d'outils graphique (par exemple, pouvoir associer les rôles aux outils, des outils utilisant des mécanismes d'introspection des classes pour l'obtention des services fournis, etc.).

D'autre part, la robustesse du prototype qui a été améliorée, qui permet un fonctionnement dans un contexte distribué.

Application en vraie grandeur

Les différentes variantes du scénario de transfert de produit nous a permis de faire une première validation de nos travaux. L'application en vraie grandeur sur des projets industriels permettra une deuxième validation pour notre proposition. Dans ce contexte, et dans la continuation de nos travaux, nous envisageons de construire une fédération d'outils dont l'application est le scénario du ticket de transport. Ce scénario est une problématique industrielle et des expérimentations devraient être effectuées dans ce cadre.

D'autre part, suivant les opportunités, des expérimentations devraient mettre en oeuvre une fondation du processus en incluant l'EGLCP APEL dans sa version 5 au sein d'une fédération.

Etude de l'autonomie des outils

Dans le cadre de cette thèse, nous avons fait l'hypothèse selon laquelle les outils sont autonomes et le restent (autonomie locale). Cependant, nous avons constaté que cette autonomie, même locale, pouvait engendrer des comportements indésirables (incohérence entre l'univers commun de l'application et l'univers/l'environnement de l'outil engendrée du fait d'actions locales de ce dernier).

L'étude de l'autonomie des outils consiste à voir en quelle mesure cette autonomie doit être préservée et, dans le cas contraire, donner au concepteur les moyens de l'inhiber. Cette inhibition passe par les façades "spécialisées" qui vont pouvoir agir (interception d'une action de l'utilisateur, etc.).

Définir un langage pour la formalisation des fédérations

Enfin, une fois que les concepts auront été entièrement validés et stabilisés (application en vraie grandeur), nous pensons définir un ou plusieurs langages (appelés langages de description de fédérations d'outils – LDFO) de plus haut niveau pour la description et la formalisation des fédérations d'outils ; un langage uniquement descriptif (cas du langage FTML) ne peut couvrir l'expression de la coordination au niveau des aspects. Une autre possibilité consisterait à étudier et à évaluer un certain nombres de langages déjà disponibles pour la définition des fédérations d'outils.

Dans ce cas, un ou plusieurs compilateurs devront être ensuite développés ; ces compilateurs permettront, à partir d'une description de fédération d'outils exprimée à l'aide du ou des langages définis (LDFO), de générer les exécutables de la fondation de contrôle.

Une autre alternative (celle qui a été choisie actuellement) consiste à stocker le modèle de fédération d'outils au niveau de l'univers commun de contrôle (UCC). Dans ce cas, le ou les compilateurs permettront, à partir de la description exprimée à l'aide du ou des langages (LDFO), de générer les objets du SGBDOO (l'UCC). La fondation de contrôle est alors générique et indépendante d'un modèle de fédération d'outils particulier ; les composants de la

fondation de contrôle interprètent le modèle et gèrent l'état de l'UCC (voir quatrième chapitre).

VI.4.2 Nouvelles applications

Méta-fédérations

Il s'agit d'appliquer l'approche proposée dans le cadre de cette thèse aux fédérations de fédérations, c'est-à-dire, en considérant qu'un participant d'une fédération peut être une autre fédération et de voir comment cela peut-il être géré ; c'est-à-dire, sous quelles conditions, quelles contraintes techniques, etc.

Dans le cas général, il est tout à fait possible d'étendre l'approche proposée pour la construction de fédérations à partir de fédérations (une fédération étant vue comme un outil). Soit une fédération *F1* participant à une fédération *F*. Les conditions à vérifier pour qu'une fédération participe à une autre fédération sont :

- la disponibilité d'une interface : l'interface de la fédération *F1* est celle de l'univers commun de l'application de *F1* ;
- pouvoir associer une façade à *F1* (la façade allant utiliser les mécanismes d'interaction de *F*) ;
- définir le comportement de *F1* dans *F* en utilisant les concepts présentés au cours de ce document (rôles, aspects, représentant, façade, etc.).

Sous quelles conditions les concepts de représentants, de façades peuvent s'appliquer aux fédérations (et non plus simplement aux outils) ?

L'idée serait d'associer un représentant et une façade à la fédération participante. Il faudra donc regarder si ces composants sont les composants génériques et parvenir à définir de tels composants, éventuellement spécialisés pour des fédérations.

Des expérimentations devront être menées pour vérifier la validité de l'approche concernant des fédérations de fédérations.

VI.4.3 Nouveaux thèmes de recherche

Evolution des fédérations

L'étude de l'évolution des fédérations d'outils est un des points importants des nouveaux thèmes de recherche. Cette question est à la convergence de nos travaux sur les fédérations, des travaux portant sur les architectures logicielles et les langages de description d'architecture. Il s'agirait de voir sous quelles conditions un outil de la fédération peut être remplacé par un autre, supprimé ou ajouté à la fédération et de façon dynamique.

Dans la mesure où l'univers commun de contrôle contient le modèle de fédération, l'étude de l'évolution d'une fédération d'outils va porter sur l'étude de l'évolution de l'univers commun et du modèle de fédération associé.

Dans le cadre du projet ESPRIT PIE, le prototype démontré a permis, lors de l'exécution du scénario, de remplacer un outil par un autre, de supprimer un outil ou d'en ajouter un et ce, en cours d'exécution de la fédération. Cette faisabilité technique et montrée à l'exécution n'a cependant pas fait l'objet de travaux de recherche approfondis sur les conditions à satisfaire pour permettre une suppression, un ajout, un remplacement, etc.

L'ensemble des moyens, des approches et des techniques permettant l'évolution feront l'objet de recherches futures. Entre autres, quels peuvent être les outils de la fondation dont l'application est l'évolution d'une fédération quelconque ?

Taxonomie et génération automatique des façades

L'approche que nous avons prise et la modélisation des outils en particulier nous permettent d'aider à la génération des façades des outils. Cependant, l'intervention manuelle reste encore importante.

Au vue des premières expérimentations, nous pensons que les façades peuvent être regroupées par catégories ; par exemple, des travaux sont menés actuellement pour développer une façade générique de tous les outils de type "gestionnaire de fichier". Les expérimentations en vraie grandeur que nous désirons menées devraient nous fournir des informations supplémentaires pour l'étude et la génération des façades.

Processus de développement des systèmes à base de composants

Une fois les expérimentations en vraie grandeur effectuées, nous pourrons voir comment s'inscrit la conception et la mise en oeuvre de fédérations de progiciels dans le processus de développement de systèmes à base d'outils de type COTS et de focaliser le travail sur l'identification du processus, de ses caractéristiques et des tâches qui le composent.

Gestion de la confidentialité, de la sécurité

La question de la sécurité, de la confidentialité est cruciale dans la mise en oeuvre des systèmes d'information des entreprises. Comment cette question peut-elle être prise en compte par notre approche ? Quels moyens met-on à la disposition du concepteur de fédérations d'outils pour sécuriser ses fédérations ? Autant de questions auxquelles nous sommes confrontées et qui devront faire l'objet de nos réflexions futures.

VI.5 Conclusion

L'approche que nous avons proposée et qui a été démontrée par la réalisation d'un prototype, constitue un premier pas. Les perspectives citées dans [Amiour 1999] et qui se limitaient au domaine des processus logiciels ont été, pour la plupart, abordées par nos travaux de recherche, s'affranchissant du domaine d'application, même s'il reste encore beaucoup à faire.

Comme le montre de façon assez significative la figure VI.1, les travaux présentés dans le cadre de cette thèse ne sont que le départ d'autres travaux qui vont s'intéresser à la même problématique. La question des fédérations de progiciels est suffisamment complexe, adresse suffisamment de problèmes qui font l'objet d'études par ailleurs, pour qu'elle soit le cœur d'autres investigations.

Enfin, un des objectifs, à terme, est de fournir un environnement complet permettant non seulement la modélisation et la mise en oeuvre de fédérations d'outils (déploiement) mais également d'offrir un support à leurs évolutions.

REFERENCES BIBLIOGRAPHIQUES

[Abdelmalki 1998] L. Abdelmalki, "Les entreprises face à la globalisation : Technologie, organisation et territoire", dans [Foulard 1998].

[Abts 1997] C. Abts, "COTS Software Integration Cost Modeling Study", Technical Report, University of Southern California, Center for Software Engineering, Barry Boehm, Director, 1997.

[Abts et Boehm 1997] C. Abts, B.W. Boehm, "COTS/NDI Software Integration Cost Estimation & USC-CSE COTS Integration Cost Calculator V2.0 User Guide", revision 1.0, University of Southern California, Center for Software Engineering, septembre 1997, http://sunset.usc.edu/research/COCOTS/index.html.

[Active 1997] Active Group home page [online], http://www.activex.org/, 1997.

[Allen et Garlan 1997] R. Allen, D. Garlan, "A Formal Basis for Architectural Connection", ACM Transactions on Software Engineering and Methodology (TOSEM), juillet 1997.

[Allen et al. 1997] R. Allen, R. Douence, D. Garlan, "Specifying Dynamism in Software Architectures", in Proceedings of Foundations of Component-based Systems Workshop, 1997.

[Allouche 1998] M. Allouche, "Une société d'agents temporels pour la supervision de systèmes industriels", thèse de doctorat, Ecole Nationale Supérieure des Mines de Saint-Etienne, Université Jean Monnet, France, 1998.

[Alloui 1996] I. Alloui, "Peace+ : un formalisme et un système pour la coopération dans les environnements de génie logiciel centrés processus ; une approche intentionnelle des interactions dans un univers multi-agent", Thèse de Doctorat présentée à l'Université Pierre Mendes France, Grenoble II, septembre 1996.

[Alloui et al. 1994] I. Alloui, S. Arbaoui, F. Oquendo " Process Centred Environments : Support for Human-Environment Interation and Environment-Mediated Human Cooperation ", *Proceedings of the 9th International Sofware Process Workshop ISPW9*, Airlie, Virginie, IEEE Computer Society Press, septembre 1994.

[Alloui et Oquendo 1996a] I. Alloui, F. Oquendo " Peace+ : a multi-agent system for computer-supported cooperative work in Software Process Centred Environments ", *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering SEKE'96*, Névada (EU), juin 1996

[Alloui et Oquendo 1996b] I. Alloui, F. Oquendo " Process Centred Environments for Software Product Lines : requirements and proposal for cooperation support ", *Proceedings of the 10th International Sofware Process Workshop ISPW10,* Ermitage du Frère Joseph, France, IEEE Computer Society Press, juin 1996.

[Alloui 1998] I. Alloui, "Requirements and approach for change support and interface definition", *Technical Report, CS-DPRO,* PIE ESPRIT LTR IV Project No. 24840, mai 1998.

[Alloui et al. 1999] I. Alloui, S. Cimpan, F. Oquendo, H. Verjus : " ALLIANCE: A Software Framework for Software-intensive Process Modeling, Enactment and Fuzzy Control ", *Proceedings of the Integrated Design and Process Science IDPT'99,* tenue conjointement avec IDPT'2000 au Texas (EU), juin 2000.

[Alloui et al. 2000] I. Alloui,S. Beydeda, S. Cîmpan, V. Gruhn, F. Oquendo, C. Schneider, "Advanced Services for Process Evolution: Monitoring and Decision Support", In Proceedings of the 7th European Workshop on Software Process Technology, EWSPT'2000, Kaprun, Autriche, février 2000.

[AMA 1991] "21st Manufacturing Entreprise Strategy", Vol.1, Lacocca Institute, University of Lehicgh, (EU), novembre 1991.

[Ambriola et al. 1990] V. Ambriola, P. Ciancarini, C. Montangero, "Software Process Enactment in Oikos", Proceedings of the 4th ACM SIGSOFT Symposium on Software Development Environments, Irvine, Californie (EU), décembre 1990.

[Ambriola et Montangero 1992] V. Ambriola, C. Montangero, "Oikos at the age of three", Proceedings of the 2nd European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norvège, septembre 1992.

[Ambriola et al. 1997] V. Ambriola, R. Conradi, A. Fuggetta, "Assessing Process-Centred Software Engineering Environments", ACM trsansactions on Software Engineering and Methodology, Vol. 6, No 3, juillet 1997, pages 283-328.

[Amiour 1997] M. Amiour, "A support for cooperation in software processes", CaiSE'97, Doctoral Symposium, Barcelone, Espagne, 16-17 juin 1997.

[Amiour et Estublier 1997] M. Amiour, J. Estublier, "Un formalisme pour le support à la coopération dans les procédés logiciels", International Conference on Software Engineering and its Application, GL'97, Paris, France, 1997.

[Amiour et al. 1997] M. Amiour, S. Dami, J. Estublier, "APEL V4: Formalism and Environment Specification", Technical Report, Adele Team, LSR Laboratory, Grenoble, France, juin 1997.

[Amiour 1999] M. Amiour, "Vers une fédération de composants interopérables pour les environnements centrés procédés logiciels", Thèse de doctorat, Université Joseph Fourier – Grenoble I, juin 1999.

[Amiour et Estublier 1998a] M. Amiour, J. Estublier " A Support for Communication in Software Processes ", *Proceedings of the 10th International Conference on Software Engineering and Knowledge Engineering SEKE'98,* San Francisco, Etats-Unis, juin 1998.

[Amiour et Estublier 1998b] M. Amiour, J. Estublier, "PIE Interoperability Support: An Approach Based on a Federation of Heterogeneous and Interoperable Components", *Deliverable IS-DDOC,* PIE ESPRIT LTR IV Project No. 24840, mai 1998.

[Anderson 1989] E. Anderson, "A Heuristic for Software Evaluation and Selection", Software Practice and Experience, Vol. 19, No. 8, août 1989, pp. 707-717.

[Andreoli et al. 1998] J.M. Andreoli, F. Pacull, D. Pagani, R. Pareschi, "Multiparty Negociation of Dynamic Distributed Object Services", In: Journal of Science of Computer Programming, 31, pp. 179-203, 1998.

[Andreoli et al. 1999] J.M. Andreoli, D. Arregui, F. Pacull, M. Rivière, J.Y. Vion-Dury, J. Willamowski, "CLF/Mekano: a framework for building virtual-enterprise applications", In: Proc. of EDOC'99, 27-30 septembre 1999, Manheim, Allemagne.

[Arbaoui 1993] S. Arbaoui *PEACE, Un formalisme fondé sur une logique modale pour la modélisation des processus logiciels évolutifs et non-monotones,* Thèse de Doctorat présentée à l'Université Pierre Mendes France, Grenoble II, juillet 1993

[Arbaoui et al. 1992] S. Arbaoui, S. Mouline, F. Oquendo, G. Tassart " PEACE : Describing and Managing Evolving Knowledge in the Software Process ", *Proceedings of the Second European Workshop on Software Process Technology,* Trondheim (Norvège), septembre 1992.

[Arbaoui et Oquendo 1993a] S. Arbaoui, F. Oquendo " Managing inconsistencies between process enactment and process performance states ", *Proceedings of the 8th International Software Process Workshop,* Warden (Allemagne), mars 1993.

[Arbaoui et Oquendo 1993b] S. Arbaou, F. Oquendo " Software process performance in PEACE ", *Proceedings of the 6th International Conference in Software Engineering and its Applications,* Paris (France), novembre 1993.

[Arbaoui et Oquendo 1994] S. Arbaoui, F. Oquendo " PEACE : Goal-Oriented Logic-Based Formalism for Process Modelling " dans [Promoter 1994].

[Avrilionis 1996] D. Avrilionis, "Mécaniques pour l'évolution et la réutilisation des processus de production de logiciels", thèse de doctorat, Université Joseph Fourier, Grenoble, France, 1996.

[Avriolinis et al. 1996] D. Avrilionis, P.Y. Cunin, C. Fernström " OPSIS: A View Mechanism for Software Processes which Supports their Evolution and Reuse ", *Proceedings of the International Conference on Software Engineering (ICSE 18),* Berlin, Allemagne, 25–29 mars 1996, IEEE Computer Society Press.

[Bachman et al. 2000] F. Bachman, L. Bass, C. Buhman, S. Comella-Dorba, F. Long, J. Robert, R. Seacord, K. Wallnau, "Volume II: Technical Concepts of Component-Based Software Engineering", Technical Report CMU/SEI-2000-TR-008, ESC-TR-2000-007, Software Engineering Institute, mai 2000.

[Balkt et Kedia 2000] L.D. Balk, A. Kedia, "PPT: A COTS Integration Case Study", ICSE 2000, Limmerick, Irlande, 2000.

[Bandinelli et al. 1992] S. Bandinelli, A. Fuggetta, C. Ghezzi, S. Grigolli " Process Enactment in SPADE ", *Proceedings of the Second Europeean Workshop on Software Process Technology,* Trondheim (Norvège), Springer Verlag, septembre 1992

[Bandinelli et al. 1993a] S. Bandinelli, A. Fuggetta, C. Ghezzi "Process Model Evolution in the SPADE Environment", *IEEE Transactions on Software Engineering,* vol. 19, décembre 1993.

[Bandinelli et al. 1993b] S. Bandinelli, A. Fuggetta, S. Grigolli " Process Modelling-in-the-large with SLANG ", *Proceedings of the 2nd International Conference on the Software Process,* Berlin (Allemagne), février 1993.

[Bandinelli et al. 1994a] S. Bandinelli, A. Fuggetta, C. Ghezzi, L. Lavazza " SPADE : An Environment for Software Process Analysis, Design and Enactment ", dans [Promoter 1994], 1994.

[Bandinelli et al. 1994b] S. Bandinelli, E. Di Nitto, A. Fuggetta, "Policies and Mechanisms to Support Process Evolution in PSEEs", In Proceedings of the 3rd International Conference on the Software Process, Virginia (EU), octobre 1994.

[Barghouti et Krishnamurthy 1993a] N.S. Barghouti, B. Krishnamurthy, "An Open Environment for Process Modeling and Enactment", In [Schäfer 1993], pages 33-36.

[Barghouti et Krishnamurthy 1993b] N. S. Barghouti, B. Krishnamuthy, "Provence: A Process Visualization and Enactment Environment", In Proceedings of the 4th European Software Engineering Conference, Garmish-Partenkirchen, Allemagne, Springer-Verlag, pp. 451-465, septembre 1993.

[Batini et al. 1986] C. Batini, M. Lenzerini, S.B. Navathe, "A comparative analysis of methodologies for database schema integration", ACM Computing Surveys, 18(4):323-364, décembre 1986.

[Beach et Bonewell 1993] J.R. Beach, L. Bonewell, "Setting-up a successful software vendor evaluation/qualification process for 'off-the-shelve' commercial software used in medical devices", *in Proceedings of the 6th Annual IEEE Symposium on Computer-Based Medical Systems*, IEEE Computer Society, Los Alamitos, pp. 284-288, juin 1993.

[Becchi et al. 2000] M. Becchi, G. Cugola, P.Y. Cunin, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, "Architecture approach and formalization", PIE LTR ESPRIT Project 34840, Deliverable D2.05, octobre 2000.

[Beck et al. 2000] M. Beck, J. Finout, B. Westlake, "Change Management and EAI", in eAI Journal, pp. 77-79, septembre 2000.

[Belkhatir et Melo 1992] N. Belkhatir, W.L. Melo, "Tempo: a software process model based on object context behavior", In the Proceedings of the 5th International Software Process Engineering and its Applications, pages 733-742, Toulouse, France, 7-11 décembre 1992.

[Belkhatir et al. 1993] N. Belkhatir, J. Estublier, W.L. Melo, "Tempo: Enhancing O.O. paradigm for modeling software engineering processes", In Schaefer editor, Proceedings of the 8th International Software Process Process Workshop, Allemagne, IEEE CS Press, 1993.

[Belkhatir et Melo 1993] N. Belkhatir, W.L. Melo, "Tempo: Specifying Process Evolution by Object Roles and Temporal Events", Proceedings of the 2nd International Conference on the Software Process, Berlin, Allemagne, février 1993.

[Belkhatir et al. 1994] N. Belkhatir, J. Estublier, W.L. Melo, "ADELE-TEMPO: An Environment to Support Process Modelling and Enaction", dans [Promoter 1994], 1994.

[Ben-Shaul et Kaiser 1993] I.Z. Ben-Shaul, G.E. Kaiser, "Process evolution in the Marvel environment", dans [Schäfer 1993].

[Ben-Shaul et Kaiser 1995] I.Z. Ben-Shaul, G.E. Kaiser, "An Interoperability Model for Process-Centered Software Engineering Environments and its Implementation in Oz", technical Report CUCS-034-95, Columbia University Department of Computer Science, 1995.

[Ben-Shaul et Kaiser 1998] I.Z. Ben-Shaul, G.E. Kaiser, "Federating Process-Centred Environments: the Oz Experience", ASE journal, vol. 5, Issue 1, janvier 1998, Kluwer Academic Publishers.

[Bergey et al. 1997] J.K. Bergey, L.M. Northrop, D.B. Smith, "Enterprise Framework for the Disciplined Evolution of Legacy Systems", Technical Report CMU/SEI-97-TR-007, ESC-TR-97-007, Software Engineering Institute, Carnegie Mellon University, octobre 1997.

[Beydeda et Cîmpan 2001] S. Beydeda, S. Cîmpan, " Design of change support component", PIE ESPRIT Project 34840, Deliverable D4.05, février 2001.

[Boehm 1981] B.W. Boehm *Software Engineering Economics*, Prentice Hall ed., 1981

[Boehm 1986] B.W. Boehm "A Spiral Model for Software Development and Enhancement", *ACM SIGSOFT Software Engineering Notes*, vol. 11, No. 4, août 1986.

[Bolcer et Taylor 1996] G.A. Bolcer, R.N. Taylor, "Endeavours: A Process System Integration Infrastructure", 4th International Conference on Software Process ICSP4, 2-6 décembre 1996.

[Bolusset et al. 1999] T. Bolusset, F. Oquendo, H. Verjus, "Software Component-based Federations are Software Architectures too", *Proceedings of the International Process Technology Workshop, IPTW'99*, Villars-de-Lans, France, septembre 1999.

[Bolusset et al. 2000] T. Bolusset, L. Da Silva, F. Oquendo, "Développement à base de composants : une approche centrée architecture pour le raffinement et l'implémentation de logiciels en Java Beans", In Proceedings of ICSSEA'2000, CNAM, Paris, décembre 2000.

[Boudier et al. 1988] G. Boudier, R. Minot, I. M. Thomas, "An overview of PCTE and PCTE+", In Proceedings of the 3rd ACM Symposium on Software Development Environments, Boston, Massachusetts (USA), November 28630 1988. In it ACM SIGPLAN Notices, 24(2): 248-257, février 1989.

[Bourgeois 1985] J. Bourgeois, "Ateliers de Génie Logiciel: Etat de l'art et perspectives", *Génie Logiciel*, No 1, 1985.

[Boyer 1994] F. Boyer, "Coordination entre outils dans un environnement intégré de développement de logiciels", Thèse de doctorat de l'université Joseph Fourier, Grenoble I, février 1994.

[Brockschmidt 1995] K. Brockschmidt, "Inside OLE, 2nd edition", Microsoft Press, 1995.

[Brown et Wallnau 1996] A. W. Brown, K.C. Wallnau, "Engineering of Component-Based System", 7-15, Component-Based Software Engineering: Selected Papers from the Software Engineering Institute (SEI), Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Brynooghe et al. 1994] R.F. Brynooghe, R.M. Greenwood, I. Robertson, B.C. Warboys, "PADM: Towards a Total Process Modelling System", dans [Promoter 1994], 1994.

[Busse et al. 1999] S. Busse, R.D. Kutsche, U. Leser, H. Weber, "Federated Information Systems: Concepts, Terminology and Architectures", Technical Report Nr. 99-9, Technische Universität Berlin, 1999.

[Buxton et Druffel 1980] J. Buxton, L. Druffel, "Requirements for an ADA Programming Support Environment: Rationale of Stoneman", In Proceedings of the IEEE Conference on Computer Software and Applications, Chicago, Illinois (USA), octobre 1980.

[Canals et al. 1994] G. Canals, N. Boudjlida, J.-C; Derniame, C. Godart, J. Longchamp, "ALF: A Framework for Building Process-Centred Software Engineering Environments", pages 153-185. Dans [Promoter 1994], 1994.

[Carney 1997], D. Carney, "Assembling Large Systems from COTS Components. Opportunities, Caution and Complexities", SEI Monograph on Use of Commercial Software in Government Systems, SEI, Pittsburgh, EU, juin 1997.

[Casati et al. 1996] F. Casati, S. Ceri, B. Pernici, G. Pozzi, "Semantic workflow interoperability", Proceedings of the 5th International Conference on Extending Database Technology, Avignon, France, mars 1996.

[Chappell 1996a] D. Chappell, "DCE and Objects" [online], http://www.osf.org/dec/3rd-party/ChapRpt1.html, 1996.

[Chappell 1996b] D. Chappell, "Understanding ActiveX and OLE", Microsoft Press Redmond, 1996.

[Chaudet et al. 2000a] C. Chaudet, M. Greenwood, F. Oquendo, B. Warboys, "A Formal Language For Describing Evolving Software Systems: Architectural Compositionality And

Evolvability", Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP 2000) - 2nd Workshop on Object-Oriented Architectural Evolution, Sophia Antipolis, France, 12-16 juin 2000.

[Chaudet et al. 2000b] C. Chaudet, F. Oquendo, "A Formal Architecture Description Language Based on Process Algebra For Evolving Software Systems", Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00), Grenoble, France.

[Cîmpan et Oquendo 1998] S. Cîmpan, F. Oquendo "Fuzzy indicators for monitoring software processes", *Proceedings of the 6th European Workshop on Software Process Technology, EWSPT'98*, Springer Verlag, Weybridge (Angleterre), septembre 1998.

[Cîmpan et Verjus 1998a] S. Cîmpan, H. Verjus, "Requirements and approach for monitoring support and interfaces definition", *Technical Report, MS-DDOC*, PIE ESPRIT LTR IV Project No. 24840, mai 1998.

[Cîmpan et Verjus 1998b] S. Cîmpan, H. Verjus, "Requirements and approach for decision support and interfaces definition", *Technical Report, DS-DDOC*, PIE ESPRIT LTR IV Project No. 24840, May 1998.

[Cîmpan et Oquendo 2000a] S. Cîmpan, F. Oquendo, "On the Application of Fuzzy Sets Theory to the Monitoring of Software-intensive Processes", Software Process Improvement and Practice Journal, John Wiley & Sons, 2000.

[Cîmpan et Oquendo 2000b] S. Cîmpan, F. Oquendo, "OMEGA: a language and system for on-line monitoring of software processes", ACM Software Engineering Notes, juillet 2000.]

[Cîmpan 2000] S. Cîmpan, "Omega : Un formalisme et un système pour le monitoring des processus dans le cadre des environnements de génie logiciel", thèse de doctorat, Université de Savoie, janvier 2000.

[Coad et Yourdon 1991a] P. Coad, E. Yourdon, "Object Oriented Analysis", Second Edition, Yourdon Press, Prentice Hall, 1991.

[Coad et Yourdon 1991b] P. Coad, E. Yourdon, "Object Oriented Design", Yourdon Press, Prentice Hall, 1991.

[COM 1995] Microsoft Corporation, "The Component Object Model Specification, Version 0.9, October 24, 1995.

[Conradi et al. 1992] R. Conradi, C. Fernström, A. Fuggetta, R. Snowdon, "Towards a Reference Framwork for Process Concepts", presented at 2nd European Workshop on Software Process Technology, EWSPT'92, Trondheim, Norvège, J.C. Derniame editor, Springer-Verlag LNCS 635, pages 3-17, 1992.

[Conradi et al. 1994] R. Conradi, M. Hagaseth, J.O. Larsen, M.N. Nguyen, B.P. Munch, P.H. Westby, W. Zhu, M.L. Jaccheri, C. Liu, "EPOS: Object-Oriented Cooperative Process modelling", dans [Promoter 1994].

[Cugola et al. 1999] G. Cugola, P.Y. Cunin, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, "Concepts and paradigms for the design of evolving, distributed and mobile processes", PIE LTR ESPRIT Project 34840, Deliverable D2.03, novembre 1999.

[Cugola et al. 2000a] G. Cugola, P.Y. Cunin, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, "Concepts and paradigms for the design of evolving, distributed and mobile processes ", PIE LTR ESPRIT Project 34840, Deliverable D2.04, mars 2000.

[Cugola et al. 2000b] G. Cugola, P. Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, "Support for Software Federations: the PIE Platform", Proceedings of the

7th European Workshop on Software Process Technology, EWSPT'2000, Autriche, février 2000.

[Cugola et al. 2000c] G. Cugola, P.Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Riviere, H. Verjus, "Customizing the behavior of middleware: the PIE approach" in Proceedings of the Workshop on Reflective Middleware (RM2000), New York (USA), 7-8 avril 2000.

[Cunin et al. 1999] P.Y. Cunin, S. Dami, J.J. Auffret " Refinement of the PIE Workpackages " Technical Report D1.00, PIE LTR ESPRIT Project 34840, 1999.

[Cunin 2000] P.Y. Cunin, "The PIE project: An Introduction", In Proceedings of the 7th European Workshop on Software Process Technology, EWSPT'2000, Kaprun, Austria, février 2000, pages 39-54, LNCS 1780, Springer-Verlag.

[Curtis et al. 1992] B. Curtis, M. Kellner, J. Over, "Process Modelling", Communications of the ACM, vol. 35, No. 9, septembre 1992.

[Dami et al. 1995] S. Dami, P. Pouzet, J. Routin, "Internal specification of APEL v2", Esprit deliverable, PERFECT project N°9090, LGI/IMAG, Grenoble, France, mars 1995.

[Dami 1999] S. Dami " APEL v5 Preview ", Technical Report, PIE ESPRIT Project 34840, septembre 1999

[DCOM 1997] Microsoft Corporation, "Distributed Component Object Model Protocol DCOM 1.0", draft, novembre 1996.

[DeLine 1999] R. DeLine, "Avoiding Packaging Mismatch with Flexible Packaging", In ICSE'99, Proceedings of the 1999 International Conference on Software Engineering, pp. 97-106, Los Angeles CA, EU.

[De Michelis et al. 1998] G. De Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M.P. Papazoglkou, K. Pohl, J. Schmidt, C. Woo, E. Yu, "Cooperative Information Systems: a Manifesto", dans [Papazoglou et Schlageter 1998], pp. 315-363, 1998.

[Derniame et al. 1994] J.C. Derniame et al., "Life Cycle Process Support in PCIS", In Proceedings of the PCTE'94 Conference, 1994.

[Di Nitto et Fuggetta 1995] E. Di Nitto, A. Fuggetta, "Integrating process technology and CSCW", proceedings of the 4th European Workshop on Software Process Technology, LNCS 913, Springer-Verlag, avril 1995.

[Dieng et al. 2000] R. Dieng, O. Corby, A Giboin, J. Golebiowska, N. Matta, M. Ribière, "Méthodes et outils pour la gestion des connaissances", Dunod, Paris, 2000.

[Dieters et gruhn 1998] W. Dieters, V. Gruhn, "Process Management in Practice Applying the FUNSOFT Net Approach to Large-Scale Processes", Kluwer Academic Publishers, Boston 1998.

[Dowson et Fernström 1994] Dowson M., Fernstrom, C. "Towards requirements for enactment mechanisms", *Proceedings of the Third Europeean Workshop on Software Process Technology (EWSPT '94)*, LNCS 772, Springer Verlag, Villard de Lans, France, février 1994.

[DSoS 2000] Dependable Systems of Systems, "State of the Art Survey", Project funded by the European Community under the "Information Society Technology" Programme (1998-2002), Deliverable BC2, septembre 2000.

[D'Souza et Wills 1999] D. D'Souza, A.C. Wills, "Objects, Components, and Frameworks with UML: The Catalysis Approach", Boston, Ma., Addison-Weisley, 1999.

[Estublier et al. 1992] Estublier J., N. Belkhatir, M. Nacer, W. Melo "Process-Centred SEE and Adele", *Proceedings of the 5th International Workshop on Computer-Aided Software Engineering, CASE'92*, IEEE Computer Society Press, Montreal, Quebec, Canada, 1992.

[Estublier et Dami 1995] J. Estublier, S. Dami, "APEL v3 specification", Esprit deliverable, PERFECT Project, LSR/IMAG, Grenoble, France, December 1995.

[Estublier et al 1997a] J. Estublier, S. Dami, M. Amiour, "APEL, an effective software process environment", MFPE'97, Gammarth, Tunisie, 1997.

[Estublier et al. 1997b] J. Estublier, S. Dami, M. Amiour, "High Level Modeling for SCM", 7th International Workshop on Software Configuration Management (SCM7), Boston (EU), mai 1997.

[Estublier et al. 1998a] J. Estublier, S. Dami, M. Amiour "APEL: a Graphical Yet Executable Formalism for process Modelling", *ASE journal (Automated Software Engineering)*, vol. 5, Issue 1,1998 Kluwer Academi Publishers.

[Estublier et al. 1998b] J. Estublier, P.Y. Cunin, N. Belkhatir " Architectures for Process Support System Interoperability ", *Proceedings of the 5th International Conference on Software Process ICSP 5*, 1998 Chicago, Illinois, EU, juin 1998.

[Estublier et Barghouti 1998] J. Estublier, N.S. Barghouti, "Interoperability and Distribution of Process-Sensitive Systems", Software Engineering for Parallel and Distributed Systems, Kyoto, Japon 1998.

[Estublier et al. 1999] J. Estublier, M. Amiour, S. Dami, "Building a Federation of Process Support Systems", WACC'99, San-Francisco, CA, février 1999.

[Estublier et Verjus 1999] J. Estublier, H. Verjus, “Definition of the behaviour paradigms of a heterogeneous federation of evolving process components”, PIE LTR ESPRIT Project 34840, Deliverable D2.01, juin 1999.

[Estublier et Verjus 2001] J. Estublier, H. Verjus, “Definition and design of PSS elements to support evolution of heterogeneous, distributed and mobile processes in the PIE system", PIE LTR ESPRIT Project 34840, Deliverable D2.06, janvier 2001.

[Estublier et al. 2001a] J. Estublier, H. Verjus, P.Y. Cunin, "Building Software Federation", Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Las-Vegas, EU, juin 2001.

[Estublier et al. 2001b] J. Estublier, H. Verjus, P.Y. Cunin, "Modelling and Managing Software Federation", Proceedings of the Enruopean Conference on Component-Based System Engineering (CBSE'2001), (à paraître).

[Estublier et al. 2001c] J. Estublier, H. Verjus, P.Y. Cunin., "Modelling and Managing Software Federations", *European Conference on Software Engineering (ESEC)*, Wien, Austria. 10-14 septembre 2001 (à paraître).

[FAR 1996] Federal Acquisition regulations, Washington, DC: General Services Administration, 1996.

[Favrel 1998] J. Favrel, "L'entreprise virtuelle: un essai sur l'état de l'art", dans [Foulard 1998].

[Feldman 1979] S. I. Feldman, "Make – A program for maintaining computer programs", Software Practice and Experience, 1979.

[Fernstrom 1993] C. Fernstrom, "PROCESS WEAVER: Adding Process Support to Unix," Proceedings of the 2nd International Conference on the Software Process (ICSP' 2), Berlin, Germany, IEEE Press, février 1993.

[Fernstrom et Brnoisz 1993] C. Fernstrom, D. Brnoisz, "Process Support in Practice: A Case Study," Proceedings of the 6th Conference on Software Engineering Environments (SEE'93), Reading, Angleterre, IEEE Press, juillet 1993.

[FIPA 1997] Foundation for Intelligent Physical Agents, "FIPA Specification Part 3 Agent Software Integration", http://www.fipa.org, octobre 1997.

[FIPA 1998] Foundation for Intelligent Physical Agents, "FIPA Specification Part 1 Agent Management", http://www.fipa.org, octobre 1998.

[Foulard 1998] C. Foulard, "L'entreprise Communicante", Hermes, 1998.

[Fox et al. 1997] G. Fox, K. Lantner, S. Marcom, "A Software Development Process for COTS-based Information System Infrastructure", in Proceedings of the 5th International Symposium on Assessment of Software Tools, 1997.

[Francou 2001] L. Francou, "Process Server API", version 2.5.4, PIE ESPRIT Project Increment, avril 2001.

[Fuggetta 1993] A. Fuggetta, "A classification of CASE technology", IEEE Computer, 26(12):25-38, décembre 1993.

[Geppert et al. 1998] A. Geppert, M. Kradolfer, D. Tombros, "Federating Heterogeneous Workflow Systems", Technical Report 98.5, Department of Computer Science, University of Zurich, at http://www.ifi.unizh.ch/techreports/TR_1998.html, 1998.

[Godart 1993] C. Godart, "Contribution à la modélisation des procédés de fabrication de logiciel: support au travail coopératif", thèse de doctorat, Université de Nancy I, janvier 1993.

[Goldberg 1984] A. Goldberg, "Smaltalk-80 : The Interactive Programming Environment", Reading MA, Addison Weisley Publishing Company, 1984.

[Graw et Gruhn 1995] G. Graw, V. Gruhn, "Process Management in-the-many", 4th European Workshop, EWSPT'95, avril 1995.

[Gruber 1993] T. Gruber, "A translation approach to portable ontology specifications", Knowledge Acquisition, 5(2):199-220, 1993.

[Harold 2000] E. R. Harold, "XML – Le guide de l'utilisateur", éditions Osman Eyrolles Multimedia, ISBN 2-7464-0070-7, traduction de Gabriel Picarde, février 2000.

[Hayden et al. 1999] S. Hayden, C. Carrick, Q. Yang., "Architectural Design Patterns for Multiagent Coordination", *In Proceedings of the International Conference on Agent Systems '99. (Agents'99)* Seattle, WA, mai 1999.

[Heimbigner 1992] D. Heimbigner, "The ProcessWall: a Process State Server Approach to Process Programming", ACM-SDE, décembre 1992.

[Heiler et al. 1991] S. Heiler, M. Siegel, S. Zdonik, "Heterogeneous information systems: Understanding integration", in First International Workshop on Interoperability in Multi-database Systems", pages 14-21, Kyoto, Japon, avril 1991, IEEE Computer Society Press.

[Hostachy 2000] E. Hostachy, "Le système d'information doit être centré sur l'EAI", Informatiques Magazine, pp. 40-44, mai 2000.

[Huff 1996] K. E. Huff, "Software Process Modelling", Trends in Software Process, pages 1-24, A. Fuggetta et A. Wolf ed., Wiley and Sons Ltd, 1996.

[Humphrey 1991] S. Humphrey "Key Practices of the Capability Maturity Model". SEI, 1991.

[ICSSEA 2001] Appel à communications, "International Conference on System and Software Engineering and its Applications", Paris, 2001, www.cnam.fr/CMSL.

[IEEE 1990] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[InfoMag 2000] Informatiques Magazine, "EAI – Du middleware au workflow de processus", dossier, pp. 48-56, novembre 2000.

[Inoue et al. 1989] K. Inoue, T. Ogihara, T. Kikuno, K. Torii, "A Formal Adaptation Method for Process Description", *In Proceedings of the 11th International Conference on Software Engineering*, 1989.

[Jaccheri et al. 1998] M.L. Jaccheri, G.P. Picco, P. Lago, "Eliciting Software Process Models with the E3 Language", TOSEM 7(4):368-410, 1998.

[Javasoft 1997] Javasoft, "Java beans specification 1.0 a", http://splash.javasoft.com/beans/beans.100A.pdf, 1997.

[Lutz 2000] J. C. Lutz, "EAI Architecture Patterns", EAI Journal, mars 2000, p. 67-73.

[Jung et al. 1999] Jung, Ho-Won, Byoungju Choi, "Optimization models for quality and cost of modular software systems", European Journal of Operational Research, vol. 112, No. 3, pp. 613-619, février 1999.

[Junkermann et al. 1994] G. Junkermann, B. Peuchel, W. Schaefer, S. Wolf, "Merlin: Supporting Cooperation in Software Development Through a Knowledge-Based Environment", dans [Promoter 1994], 1994.

[Kaiser et al. 1988] G. Kaiser, P. Feiler, S. Popovich "Inteligent Assistance for Software Development and Maintenance", *IEEE Software*, mai 1988.

[Kontio 1996] J.A. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection", in Proceedings of the 18th International Conference on Software Engineering, mars 1996.

[Krishnamurthy et al. 1991] R. Krishnamurthy, W. Litwin, W. Kent, Language Features for Interoperability of Databases with Schematic Discripancies", ACM SIGMOD International Conference on Management of Data, Denver, Colorado, pp. 40-49, 1991.

[Kruchten 1999] P. Kruchten, "The Rational Unified Process: An Introduction", Addison-Wesley, 1999.

[Kuvaja 1994] P. Kuvaja *Software Process assesment & improvement : The bootstrap approach*. Blackwell, Oxford, Angleterre, 1994

[Latrous 1997] S. Latrous *Gestion de l'évolution des processus en génie logiciel : une approche par contraintes dans un environnement multi-agent réflexif*, Thèse de Doctorat présentée à l'Université Pierre Mendes France, Grenoble II, avril 1997.

[Latrous et Oquendo 1995] S. Latrous, F. Oquendo " PEACE+/PECAM : une machine virtuelle pour l'exécution et l'évolution des processus logiciels ", *Proceedings of the 8th International Conference on Software Engineering and its Applications GL'95*, Paris, France, novembre 1995

[Latrous et Oquendo 1996] S. Latrous, F. Oquendo " A Reflective Multi-Agent System for Software Process Enaction and Evolution ", *Proceedings of the 1st International Conference of Practical Applications of Multi-agents Systems PAAM96*, Londres, RU, avril 1996.

[Lee et Benford 1995] O.-K. Lee, S. Benford, "An explorative model for federated trading in distributed computing environments", In Proceedings of the International Conference on Open Distributed Processing, février 1995.

[Lerner et al. 1998] B. Staudt Lerner, L.J. Osterweil, S.M. Sutton Jr., A.E. Wise, "Programming Process Coordination in Little-JIL", Volker Gruhn (Ed.): Software Process

Technology, 6th European Workshop, EWSPT '98, Weybridge, UK, September 16-18, 1998, Proceedings. Lecture Notes in Computer Science, Vol. 1487, Springer, 1998, ISBN 3-540-64956-5.

[Long 1989] F. Long editor, Proceedings of International Workshop on Software Engineering Environments", volume 467 of LNCS, Chinon, France, September 18-20, 1989, Springer-Verlag, Berlin, Allemagne, 1990.

[Lomgchamp et al. 1992] J. Longchamp, C. Godart, J.C. Derniame, "Les environnements intégrés de production de logiciel", technique et science informatique, vol. 11-n°1, 1992.

[Longchamp 1993] J. Longchamp, "A structured conceptual and terminological framework for software process engineering", In Proceedings of the 2nd International Conference on the Software Process, pages 41-53, IEEE Computer Society Press, février 1993.

[Lutz 2000] J.C. Lutz, "EAI Architecture Patterns", eAI Journal, pp. 64-73, mars 2000.

[Madhavji 1991] N. M. Madhavji, "The Process Cycle", *Software Engineering Journal*, septembre 1991.

[Magee et al. 1995] J. Magee, N. Dulay, S. Eisenbach, J. Kramer, "Specifying Distributed Software Architectures", in Proceedings of the 5th European Software Engineering Conference (ESEC), LNCS 989, pp. 127-153, 1995.

[Magee et al. 1997] J. Magee, N. Tseng, J. Kramer, "Composing Distributed Objects in CORBA", in Proceedings of the International Symposium on Autonomous Decentralized Systems (ISADS), 1997.

[Malone 1997] T. W. Malone, "Modeling coordination in organizations and markets", Management Science 33(10): 1317-1332 (also published in Readings in Distributed Artificial Intelligence, A.H. Bond, L. Gasser, editors, pages 227-234, Morgan Kaufmann, 1988).

[Malone et al. 1988] T.W. Malone, R.E. Fikes, M.T. Howard, "Enterprise: A marcket-like task scheduler for distributed computing environments", in B.A. Huberman, editor, The Ecology of Computation, North-Holland Publishing Company, Amsterdam, 1988.

[Malville et Bourdon 1998] E. Malville, F. Bourdon, "Task Allocation: A Group Self-Desing Approach", In the Proceedings of the 3rd International Conference on Multi-Agents Systems (ICMAS'98), Paris, juillet 1998.

[Malville 1999] E. Malville, "L'auto-organisation de groupes pour l'allocation de tâches dans les Systèmes Multi-Agents : Application à CORBA", thèse de doctorat, Université de Savoie, France, mars 1999.

[Mann 1999] J.E. Mann, "Workflow and EAI", in eAI Journal, pp. 49-53, September/October 1999.

[Marvie 1999] R. Marvie, "CORBA Components: la proposition unifiée – Du modèle d'objets au modèle de composants", Université de Lille, France, mai 1999.

[Marvie et al. 2000] R. Marvie, P. Merle, J.M. Geib, "Towards a Dynamic CORBA Component Platform", the 2nd Symposium of Distributed Objects & Applications, Antwerp, Belgium, septembre 2000.

[Medvidovic et Rosenblum 1997] N. Medvidovic, D. Rosenblum, "Domains of Concern in Software Architectures and Architecture Description Languages", in Proceedings of the USENIX Conference on Domain-Specific Languages, October 15-17, Santa-Barbara, CA, EU, 1997.

[Medvidovic et al. 1999] N. Medvidovic, D.S. Rosenblum, R.N. Taylor, "A Language and Environment for Architecture-based Software Development and Evolution", in Proceedings of the 21st International Conference on Software Engineering (ICSE), pp. 44-53, 1999.

[Medvidovic et Taylor 2000] N. Medvidovic, R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, janvier 2000.

[Melo 1993] W.L. Melo , "TEMPO : Un Environnement de Développement de Logiciel Centré Procédés de Fabrication", Thèse de Doctorat présentée à l'Université Joseph Fourier-Grenoble I, 1993.

[MESA 1997a] MESA International, "Execution-Driven Manufacturing Management for Competitive Advantage", White Paper Number 5, http://www.mesa.org, 1997.

[MESA 1997b] MESA International, "MES Functionalities & MRP to MES Data Flow Possibilities", White Paper Number 2, http://www.mesa.org, mars 1997.

[Miller et al. 1996] J.A. Miller, A. Seth, K.J. Kochut, X. Wang, "CORBA-Based Run-Time Architectures for Workflow Management Systems", Journal of Database Management, Special Issue on Multidatabases, 7(1):16-27, 1996.

[Miller 1998] R.J. Miller, "Using Schematically Heterogeneous Structures", in: L.M. Haas, A. Tiwari, ACM SIGMOD International Conference on Management of Data, Seattle, Washinghton (EU), pp. 189-200, 1998.

[Minsky 1991] Naftaly H. Minsky, "The imposition of protocols over open distributed systems", IEEE Trans. Software Engineering, pages 183--195, février 1991.

[Minsky et Ungureanu 1997] N.H. Minsky, V. Ungureanu, "Regulated Coordination in Open Distributed Systems", In David Garlan, Daniel Le Metayer, editors, Proceedings of Coordination'97: Second International Conference on Coordination Models and Languages; LNCS 1282, pages 81-88, septembre 1997.

[Montangero et Ambriola 1994] C. Montangero, V. Ambriola " OIKOS : Constructing Process-Centered SDEs ", dans [Promoter 1994]

[Moore 1988] R.E. Moore " Autoepistemic logic ", *Non-Standard Logics for Automated Reasoning*, P. Smets, A. Mamdani, D. Dubois et H. Prade (Eds.), Academic Press, Londre, 1988.

[Morisio et al. 2000] M. Morisio, C. Seaman, A. Parra, V. Basili, S. Condon, S. Kraft, "Investigating and Improving a COTS-Based Software Development Process", Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Irlande, juin 2000.

[Morton 1991] M.S. Morton, editor, "The Corporation of the 1990s: Information Technology and Organisational Transformation", Oxford University Press, 1991.

[Morton 1995] M.S. Morton, "L'entreprise compétitive au futur: technologies de l'information et transformation de l'organisation", les éditions d'organisations, 1995.

[Mularz 1995] D.E. Mularz, "Pattern-based integration architectures", Chapter 7 in James O. Coplein, Douglas C. Schmidt, editors, Pattern Languages of Program Design, 1995. Addison-Wesley.

[Muller 1996] P.A. Muller, "Modélisation Objet avec UML", Editions Eyrolles, 1996.

[Oberndorf 1988] P.A. Oberndorf, "The common Ada programming support environment (APSE) interface set (CAIS)", IEEE Transactions on Software Engineering, 14(6), juin 1988.

[ODP 1995] "ITU-T X.901 ¦ ISO/IEC 10746-1 ODP Reference Model Part. 1 Overview", SC21 N8926rev, Draft Interntional Standard output from the editing meeting in Helsinki (Finlande), 15-18 mai 1995.

[OMG 1991] Object Management Group, " The Common Object Request Broker: Architecture and Specification ", TC Document Number 91.12.1, Revision 1.1., 1991.

[OMG 1997a] Object Management Group "CORBA services : Common Object Services Specification", USA, November 1997. OMG TC Document formal/98-07-05 http://www.omg.org/corba/sectrans.htm

[OMG 1997b] Object Management Group, "UML Notation Guide", 1.1 Edition, 1997.

[OMG 1997c] Object Management Group, "UML Semantics", 1.1 Edition, 1997.

[OMG 1997d] Object Management Group, "Object Constraint Language Specification", 1.1 Edition, 1997.

[OMG 1997e] Object Management Group, "Meta Object Facility (MOF) Specification", Joint revised Submission, OMG Document ad/97-08-14, 1997.

[OMG 1998] Object Management Group, "The Common Object Request Broker : Architecture and Specification, revision 2.2" OMG, USA, février 1998. OMG TC Document formal/98-07-01 http://www.omg.org/corba/corbiiop.htm.

[OMG 2000] Object Management Group, "CORBA Security Services Specification", v. 1.5, Inc, 2000, http://www.omg.org/technology/documents/formal/security_service.htm.

[Oquendo et al. 1991] F. Oquendo, C. Boudier, F. Callo, R. Minot, I. Thomas, "The PCTE+'s OMS: A Distributed Software Engineering Database System for Supporting Large-Scale Software Development Environments", *Proceedings of the 2nd International Symposium on Database Systems for Advanced Applications*, Tokyo, Japon, avril 1991.

[Oquendo 1995] F. Oquendo, " SCALE: Process Modelling Formalism and Environment Framework for Goal-directed Cooperative Processes ", Proceedings of the Seventh *International Conference on Software Engineering Environments (SEE'95)*, Noordwijkerhout, Pays-Bas, avril 1995. IEEE Computer Society Press.

[Orfali et al. 1996] R. Orfali, D. Harkey, J. Edwards, "Objets répartis – Guide de survie", International Thomson Publishing, ISBN 2-84180-002-4, Paris, 1996.

[OSF 1996] Open Software Foundation, "The OSF Distributed Computing Environment" [online], http://www.osf.org/dce/, 1996.

[Osterweil 1987] L. Osterweil "Software Processes are Software Too", *Proceedings of the 9th International Conference on Software Engineering*, Monterey, mars 1987

[Özsu et Valduriez 1999] M.T. Özsu, P. Valduriez, "Principles of distributed database systems", 2nd edition, Prentice Hall, 1999.

[Papazoglou et Schlageter 1998] M.P. Papazoglou, G. Schlageter (eds.), "Cooperative Information Systems – Trends and Directions", Academic Press, 1998.

[Patzer 2000] A. Patzer, "Programmation Java côté serveur", traduit de l'anglais, édition Eyrolles, ISBN 2-212-09109-5, Paris, 2000.

[Paulk et al. 1991] M.C. Paulk, B. Curtis, M.B. Chrissis et coll. *Capability Maturity Model for Software*. Software Engineering Institute, CMU/SEI-91-TR-24, ADA240603, août 1991

[Paulk et al. 1993] M.C. Paulk, B. Curtis, M.B. Chrissis, C.V. Weber. *Capability Maturity Model for Software, Version 1.1*. Software Engineering Institute, CMU/SEI-93-TR-24, ESC-TR-93-177, février 1993

[PCTE 1993] ECMA Standard 149, "Portable Common Tool Environment (PCTE): Abstract Specification", 2nd Edition, European Computer Manufacturers Association (ECMA), juin 1993.

[PCTE 1994] ISO Standard 13719, "Portable Common Tool Environment (PCTE): Abstract Specification, International Standards Organization, 1994.

[Perry et Wolf 1992] D. Perry, A. Wolf, "Foundations for the Study of Software Architecture", in Software Engineering Notes, 17(4), pp. 40-52, 1992.

[PIE 1999] PIE Strategic Board *Scope of the first prototype, Version 1.1*, PIE ESPRIT LTR IV Project No. 34840, juillet 1999

[Promoter 1999] *Software Process : Principles, Methodology, Technology*. J.C. Derniame, A.B. Kaba, D. Wastell (Eds), Springer-Verlag 1999, LNCS 1500, ISBN 3-540-65516-6199

[Promoter 1994] *Software Process Modelling and Technology*, A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Research Studien Press Limited (J. Willey), 1994

[Reix 1986] R. Reix, "Processus d'informatisation et conception de l'organisation", Economie et société, 1986.

[Reix 1995] R. Reix, "Système d'informatisation et management des organisations", Vuibert 1995.

[Robert 1989] "Le petit Robert", deuxième édition, Les dictionnaires Le Robert, Paris, 1989.

[Robertson 1998] I. Robertson " Evolution Strategy Support ", *Technical Report ES-DDOC*, PIE ESPRIT LTR IV Project No. 24840, 1998.

[Rout 1995] T. Rout : "SPICE : A Framework for Software Process Assesment". *Software Process : Improvement and Practice*, 1(1), 1995

[Royce 1970] W. Royce, "Managing the development of large software systems", In Proceedings of WESTON, San Francisco, 1970.

[Rumbaugh et al. 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice Hall, 1991.

[Saltor et al. 1996] Saltor, F., Campderrich, B., Rodrguez, E., Rodriguez, L., "On Schema Levels for Federated DB Systems", In Yetongnon & Hariri (eds.): Proc. of the ISCA Int'l. Conf. on Parallel and Distributed Computing Systems (Dijon, 1996), ISCA, 766-771

[Sandoval 1995] V. Sandoval, "Les autoroutes de l'information", Hermes, Paris, 1995.

[Schäfer 1993] W. Schäfer, editor. *8th International Software Process Workshop*, IEEE Computer Society Press, 1993.

[Schill 1993] A. Schill, "DCE-The OSF Distributed Computing Environment Client/Server Model and Beyond", 283. International DCE Workshop, Karlsruhe, Germany, October 7-8, 1993. Berlin, Germany, Springer-Verlag, 1993.

[Schmidt 2000] J. Schmidt, "Enabling Next-Generation Enterprises", EAI Journal, pp. 74-80, juillet/août 2000.

[SEI 1997] Software Engineering Institute, "Using COTS Technology in your system", presentation, Carnegie Mellon University, août 1997.

[SEI 2001] Software Engineering Institute, "COCOTS – Constructive COTS", [online], http://sunset.usc.edu/research/COCOTS/cocots_main.html, février 2001 (dernière mise à jour).

[Shaw et Garlan 1996] M. Shaw, D. Garlan, "Software Architectures: Perspectives on an Emerging Discipline", Prentice Hall, 1996.

[Shaw et al. 1997] M. Shaw, R. DeLine, D.V. Klein, T.L. Ross, D.M. Young, G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them", in IEEE Transactions on Software Engineering (TSE), 21(4), pp. 314-335, 1995.

[Sheth et Larson 1990] A.P. Sheth, J.A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, Vol. 22, No. 3, pp. 183-236, septembre 1990.

[Sichman et al. 1994] J. Sichman, R. Conte, Y. Demazeau, C. Castelfranchi, "A social reasoning mechanism based on dependence networks", In Proceedings of the 11th European Conference on Artificial Intelligence, T. Cohn editor, pp. 188-192, Amsterdam, Pays Bas, août 1994.

[Sichman et Demazeau 1994] J. Sichman, Y. Demazeau, "A first attempt to use dependence situations as a decision criterion for choosing partners in multi-agent systems", In Proceedings of ECAI'94 Workshop on Decision Theory for DAI Application, Amsterdam, Pays Bas, août 1994.

[Sichman 1995] J. Sichman, "Du Raisonnement Social Chez les Agents: Une approche Fondée sur la Théorie de la Dépendance", thèse de doctorat, Institut National Polytechnique de Grenoble, France, 1995.

[Smith 1980] R.G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE transactions on computers Vol. C-29, No. 12 (December 1980), 1104-1113.

[Snowdon et Warboys 1994] R.A. Snowdon, B.C. Warboys, "An introduction to process-Centered Environments", dans [Promoter 1994].

[Soley et Stone 1995], R.M. Soley, C.M. Stone, "Object Management Architecture Guide, revision 3.0", Object Management Group, Inc. et John Wiley & Sons, Inc., EU, juin 1995, OMG TC Document ab/97-05-05. ISBN : 0-471-14193-3.

[Spaccapietra et al. 1992] S. Spaccapietra, C. Parent, Y. Dupont, "Model independent assertions for integration of heterogeneous schemas", VLDB Journal, 1(1):81-126, 1992.

[Sommerville 1989] I. Sommerville, "Software Engineering (3rd edition)", Addison-Wesley, Wokingham, 1989.

[Stonebraker et al. 1994a] M. Stonebraker, P.M. Aoki, W. Litwin, M. Olson, "Mariposa: A New Architecture for Distributed Data", in Proceedings of the 10th International Conference on Data Engineering, Houston, Texas, février 1994.

[Stonebraker et al. 1994b] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Sah, C. Staelin, "An Economic Paradigm for Query Processing and Data Migration in Mariposa", Sequoi 2000 Technical Report 94/49, University of California, Berkeley, avril 1994.

[Stonebraker 1999] M. Stonebraker, "Integrating Islands of Information", in eAI Journal, pp. 1-5, septembre/octobre 1999.

[Sun 1997] Sun Microsystems, “Java Remote Method Invocation Specification”, 10 février 1997.

[Sun 1998] Sun Microsystems, "Java Message Service", Version 1.0.5, 5 octobre 1998.

[Sun 1999] "Enterprise Java Beans Specification", Sun Microsystem, http://java.sun.com/products/ejb, 1999.

[Sutton et al. 1995a], S.M. Sutton, Jr.P.L. Tarr, L.J. Osterweil, "An Analysis of Process Languages", Technical Report 95-78, Computer Science Department, University of Massachussets, 1995.

[Sutton et al. 1995b] S.M. Sutton, D. Heimbigner, L.J. Osterweil, "APPL/A: A language for software process programming", ACM Trans. Software. Engineering and Methodology. A, juillet 1995.

[Sutton et al. 1997] S.M. Sutton, B.S. Lerner, L.J. Osterweil, "Experience Using the JIL Process Programming Language to Specify Design Processes", Technical Report 97-68, Department of Computer Science, University of Massachussetts at Amherst, 6 septembre, 1997.

[Sutton et Osterweil 1997] S.M. Sutton, L.J. Osterweil, "The Design of a Next-Generation Process Language", Proceedings of the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich, Suisse, septembre 1997.

[Suzuki et Katayama 1991] M. Suzuki, T. Katayama, "Meta operations in the process model HFSP for the dynamics and flexibility of software processes", In. Proceedings of the 1st International Conference on the Software Process, IEEE Computer Society, 1991.

[Taylor et al. 1988] R.N. Taylor, F.C. Belz, L.A. Clarke, L. Osterweil, R.W. Selby, J.C. Wileden, A.L. Wolf, M. Young " Foundation of the Arcadia Environment ", *Proceesings of ACM SIGSOFT '88 : Third Symposium on Software Development Environments*, pp. 1-13, Boston, novembre 1988.

[Teitleman et Masinter 1984] W. Teitleman, L. Masinter, "The Interlisp programming environment", In Interactive Programming Environments. D.R. Barstow, H.E. Shrobe, E. Sandewall ed. New York: McGraw-Hill, 1984.

[Tiako 1998] P.F. Tiako, "Modelling the Federation of Process Sensitive Engineering Environments: Basic Concepts and Perspectives", In Proceedings of the 6th European Workshop on the Software Process Technology, EWSPT'98, Weybridge, Angleterre, septembre 1998.

[Tiako 1999] P.F. Tiako, "Modélisation de la Fédération des Environnements Sensibles aux Procédés de Développement de Logiciels", thèse de doctorat, Institut National Polytechnique de Lorraine, juin 1999.

[Thomas 1989] I. Thomas, "PCTE interfaces: supporting tools in software engineering environments", IEEE Software, 6(6): 15-23, novembre 1989.

[Tombros et Geppert 2000] D. Tombros, A. Geppert, "Building Extensible Workflow Systems using an Event-Based Infrastructure", Wangler, L. Bergman (eds.): *Proc. 12th Int'l Conf. on Advanced Information Systems Engineering*, Lecture Notes in Computer Science, Vol 1789, Springer Verlag, Heidelberg, Allemagne, pages 325-339, Stockholm, Suède, juin 2000.

[Totland et Conradi 1995] T. Totland, R. Conradi, "A Survey and Classification of Some Research Areas relevant to Software Process Modeling", *Software Process Technology, 4th European Workshop, EWSPT '95*, Noordwijkerhout, Pays Bas, 3-5 avril 1995, Proceedings. Lecture Notes in Computer Science, Vol. 913, Springer, ISBN 3-540-59205-9.

[Uschold et Gruninger 1996] M. Unschold, M. Gruninger, "Ontologies: principles methods and applications knowledge", Engineering Review, Vol. 11, No 2, juin 1996.

[Valente et Breuker 1996] A. Valente, J. Breuker, "Towards principled core ontologies", In. B. Gaines, M. Musen (ed.), Proceedings of the 10th Banss Knowledge acquisition for Knowlegde-based systems worksshop (KAW'96), Banss, Canada, 1996, pp. 33-1, 33-20. http://ksi.cpsc.ucalagary.ca/kaw/kaw96/kwa96proc.html.

[Valetto et Kaiser 1995] G. Valetto, G.E. Kaiser, "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," 40-48. Proceedings of 7th IEEE

International Workshop on CASE. Toronto, Ontario, Canada, 10-14 juillet 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Verjus et Cîmpan 1998] H. Verjus, S. Cîmpan " Technical Description of the Monitoring and Decision Support PIE Component Prototypes ", *Technical Report, MS-DPRO & DS-DPRO*, PIE ESPRIT LTR IV Project No. 24840, avril 1998.

[Verjus 1998a] H. Verjus, "Vers une Fédération d'Environnements Centrés Processus", Journées Jeunes Doctorants, MATIS'98, Archamps, France, mars 1998.

[Verjus 1998b] H. Verjus, "Interopérabilité: état de l'art, mécanismes, travaux et orientations", Rapport technique N°0000601, 20/12/1998, LLP/CESALP Université de Savoie.

[Verjus et Oquendo 1998] H. Verjus, F. Oquendo, "Fédérations d'environnements centrés processus logiciels : une approche basée composants", Proceedings *of the 11th International* Conference on Software Engineering and its Applications (GL'98), Paris, décembre 1998.

[Verjus et al. 2000] H. Verjus, F. Oquendo, C. Chevenier, "Etat de l'art sur l'échange de données informatisées : concepts, mécanismes, travaux et tendances sur l'interopérabilité", Contrat d'études, recherche et développement, Electricité de France, Echanges de données informatisées pour des outils relevant de l'ingénierie système, livrable L3.2, mars 2000.

[Wakeman et Jowett 1990] L. Wakeman, J. Jowett, "PCTE – The standard for open repositories", Prentice Hall, 1990.

[Wallnau et Feiler 1991] K.C. Wallnau, P. H. Feiler, "Tool integration and environment architecture", technical Report CMU/SEI-91-TR-11, Carnegie-Mellon University, Software Engineering Institute, mai 1991.

[Wang et al. 2000] N. Wang, D.C. Schmidt, C. O'Ryan, "Overview of the CORBA Component Model".

[Wasserman 1989] A.I. Wasserman, "Tool Integration", In [Long 1989], 1989.

[WfMC 1995] Workflow Management Coalition, Coalition Overview, Sept. 1995, http://www.aiai.ed.ac.uk/WfMC/overview.html

[WfMC 1996a] The Workflow Management Coalition Specification, D. Hollingsworth, "The Workflow reference model", Abstract Specification, Document Number WFMC-TC-1003, Workflow Management Coalition, novembre 1996.

[WfMC 1996b] WFMC. "WPDL". Document WFMC TC-0020 , Workflow Management Coalition, 1996.

[Wiederhold 1993] G. Wiederhold, "Intelligent Integration of Information", SIGMOD Record, Vol. 22, No. 2, pp. 434-437, ACM, juin 1993.

[Wills 1999] A. Wills, "Modeling for Component Based System with Catalysis", Tutorial notes, Enterprise Distributed Object Computing (EDOC), Manheim, Allemagne, 1999.

[Wolisz et Taschammer] A. Wolisz, V. Tschammer, "Performance aspects of trading in open distributed systems", Computer Communications, pages 277-287, mai 1993.

[Woods et Barbacci 1999] S.G. Woods, M.R. Barbacci, "Architectural Evaluation of Collaborative Agent-Based Systems", Software Engineering Institute, Technical Report CMU/SEI-99-TR-025, ESC-TR-99-025, octobre 1999.

[Zisman et Kramer 1995] A. Zisman, J. Kramer, "Towards Interoperability in Heterogeneous Database Systems", Imperial College Research Report No. DOC 95/11, 1995.

Annexe A : Grammaire du langage de description de fédérations d'outils sous forme de DTD

1 Introduction à XML

XML a été développé par un groupe de travail XML (*XML Working Group*) constitué à l'initiative du consortium W3C (consortium du Web).

Les objectifs de conception de XML sont les suivants :

- XML devrait pouvoir être utilisé sans difficulté sur Internet ;
- XML devrait soutenir une grande variété d'applications ;
- XML devra être compatible avec SGML ;
- Il devrait être facile d'écrire des programmes traitant les documents XML ;
- Le nombre d'options dans XML doit être réduit au minimum, idéalement à aucune ;
- Les documents XML devraient être lisibles par l'Homme et raisonnablement clairs ;
- La conception de XML devrait être formelle et concise.

2 Spécifications du langage XML

La spécification du langage est donnée à http://www.w3c.org/TR/1998/REC-xml-19980210.

Il s'agit d'une recommandation du W3C datant du 10 février 1998. La version la plus récente peut être trouvée à http://www.w3c.org/TR/1998/REC-xml.

Cette spécification s'accompagne de normes:

- Unicode et l'ISO/CEI 10646 pour les caractères,
- le RFC 1766 Internet pour les balises d'identification de langage,
- l'ISO 639 pour les codes de noms de langue,
- l'ISO 3166 pour les codes de noms de pays.

L'ensemble de ces documents fournissent toutes les informations nécessaires à la compréhension de la version 1.0 de XML et à la construction de programmes pour traiter des documents XML.

3 Intérêts pour le choix de XML

Les trois dernières années ont été marquées par le "phénomène" XML devant formalisme d'échange reconnu et adopté ou en cours d'être adopté par les plus grands éditeurs de logiciels comme au niveau des échanges qui doivent être formalisés dans les entreprises.

D'autre part, étant par nature voué à être un vecteur d'information sur le Web (interopérabilité), il s'intègre parfaitement dans la mouvance des technologies dites "nouvelles" que sont :

- les applications distribuées et technologies des objets répartis ;
- les langages d'interopérabilité ;
- les langages de programmation modernes (de nombreux interpréteurs java existent pour XML, etc.).

D'autre part, nous nous étions heurtés aux limites des langages "descriptifs"; en particulier, les zones de conversions, de translation entre les différents concepts (particulièrement ceux de l'univers commun et ceux de chacun des outils), quand cela porte sur des aspects techniques (une opération, un message, etc.) nous obligent à recourir aux structures itératives, conditions, plus généralement, aux structures de contrôle des langages de programmations de troisième et quatrième génération.

Plutôt que de développer un nouveaux langage dont la grammaire aurait induit ces structures de contrôle et dont la complexité, de prime abord, n'aurait pas diminué pour autant, il nous a paru intéressant de se baser sur les possibilités offert par XML et, entre autre, l'encapsulation de code (java en ce qui concerne les implémentations que nous avons effectuées).

4 Qu'est-ce qu'une "DTD" ?

Un langage (comme le langage FTML) est un langage de description créé grâce à XML. Un tel langage, qui correspond à des jeux de balises (tag), est défini via une DTD. Une DTD pour *Document Type Definition*, fournit la liste des éléments, des attributs, des notations et des entités que contient le document ainsi que les règles des relations qui les régissent [Harold 2000].

Une DTD peut être insérée dans le fichier du document qu'elle régit, mais elle est plus généralement associée à ce fichier via une référence d'adresse URL externe. Une DTD externe peut ainsi être partagée par plusieurs documents.

5 Les références d'entités symboliques

Ces références vous sont données par le tableau 1. Par exemple, la référence d'entité **<** symbolise le signe "inférieur à" (le chevron gauche). En effet, dans le langage XML, la présence de ce signe est interprétée comme le début d'une balise (*tag*).

Référence d'entité	Caractère symbolisé
&	**&**
<	<
>	>
"	"
'	'

Tableau 1 Les références d'entités prédéfinies de XML

L'utilisateur devra donc être conscient de ces références et leurs bonnes utilisation, notamment dans l'encapsulation des parties de code où les syntaxes des langages de programmation ont souvent recours aux symboles sus-mentionnés.

6 Les différentes DTD et leur contenu

Les fichiers qui sont listés ci-après contiennent l'ensemble des concepts et leurs descriptions manipulés dans les documents XML permettant d'établir une description de fédération d'outils selon les objectifs présentés dans le cadre de cette thèse.

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: federation.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe FEDERATION elements.
            FEDERATION is the ROOT of a federation description.

            This description allows two main possibilities:
                - the federation is uniquely described using the
                STAGE1 section; in this case, the end user is on the
                highest level of abstraction. Interpreter tools will
                do a job to produce the federation description at the
                stage2, containing descriptions of proxies and
                wrappers uniquely based on informations provided at
                the stage 1. The end-user has to complete the
                generated federation description (wrappers and
                proxies sections).
                - the federation is uniquely described using the
                STAGE2 section; this stage requires less expertise
                than the previous but is more complete. It assumes
                that the end-user is able to provide some (partial)
                proxies and wrappers descriptions. Proxies and
                Wrappers semi-generation are deduced from this
                stage description.
                - Then, developpers will complete the generated code
                allowing to provide the federation control support
                prototype.
```

```
     ============================================================ -->


<!ELEMENT FEDERATION (
                       CU-SET,
                       (STAGE1, STAGE2) | (STAGE1) | (STAGE2)
                      )>



<!ATTLIST FEDERATION beginDate  CDATA                            #IMPLIED>
<!ATTLIST FEDERATION endDate    CDATA                            #IMPLIED>
<!ATTLIST FEDERATION priority   (prohibition | right) "right"   #IMPLIED>

<!ENTITY % STAGE1               SYSTEM          "stage1.dtd">
<!ENTITY % STAGE2               SYSTEM          "stage2.dtd">



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: cu-set.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe CU-SET elements.
            It can be interesting to describe several CUs...

     ============================================================ -->


<!-- The root elements of the XML document associated with this DTD is
     the CU-SET tag which is a set of CU.
-->
<!ELEMENT CU-SET                (CU*)>
<!ATTLIST CU-SET                comment         CDATA>
<!ENTITY % CU                   SYSTEM          "cu.dtd">



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: cu.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe CU elements.
            CU is characterised by an API...

     ============================================================ -->

<!ELEMENT CU (CU-DESC?, FEDERAL-SERVICES)>
```

```
<!ATTLIST CU cuID   ID  #REQUIRED>

<!ELEMENT CU-DESC (#PCDATA)>

<!ELEMENT FEDERAL-SERVICES (OPERATION | MESSAGE)*>

<!ENTITY % OPERATION SYSTEM "operation.dtd">

<!ENTITY % MESSAGE SYSTEM "message.dtd">



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: stage1.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe STAGE1 elements.

     ============================================================ -->


<!ELEMENT STAGE1 (
                       MODELS,
                       SITES,
                       INSTANCES,
                       ASPECTS?,
                       ROLES?,
                       ROLE-COOPERATION?,
                       )>


<!ENTITY % MODELS               SYSTEM          "models.dtd">
<!ENTITY % SITES                SYSTEM          "sites.dtd">
<!ENTITY % INSTANCES            SYSTEM          "instances.dtd">
<!ENTITY % ASPECTS              SYSTEM          "aspects.dtd">
<!ENTITY % ROLES                SYSTEM          "roles.dtd">
<!ENTITY % ROLE-COOPERATION     SYSTEM          "role_cooperation.dtd">



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: stage2.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe STAGE2 elements.

     ============================================================ -->
```

```
<!ELEMENT STAGE2 (
                      MODELS,
                      SITES,
                      INSTANCES,
                      ASPECTS?,
                      ROLES?,
                      ROLE-COOPERATION?,
                      PROXIES?,
                      WRAPPERS?
                      )>

<!ENTITY % MODELS               SYSTEM          "models.dtd">
<!ENTITY % SITES                SYSTEM          "sites.dtd">
<!ENTITY % INSTANCES            SYSTEM          "instances.dtd">
<!ENTITY % ASPECTS              SYSTEM          "aspects.dtd">
<!ENTITY % ROLES                SYSTEM          "roles.dtd">
<!ENTITY % ROLE-COOPERATION     SYSTEM          "role_cooperation.dtd">
<!ENTITY % WRAPPERS             SYSTEM          "wrappers.dtd">
<!ENTITY % PROXIES              SYSTEM          "proxies.dtd">
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: common.dtd

        Document Type Definition
        ------------------------

        Description:
            This dtd file contains usefull elements description
            that are used somewhere else.


     ============================================================ -->

<!--  -->
<!ELEMENT NAME #PCDATA>
<!ATTLIST NAME xml:space (default|preserve) "preserve">

<!ELEMENT CODE-BODY     (CODE-PIECE | CODE-FILE)>
<!ATTLIST CODE-BODY     xml:space (default|preserve) "preserve">

<!ELEMENT CODE-PIECE    (#PCDATA)>
<!ATTLIST CODE-PIECE     xml:space (default|preserve) "preserve">

<!ELEMENT CODE-FILE     (#PCDATA)>
<!-- The code body is dedicated to provide java programming language
        control structures the end-user may use.
        For example, all mapping parts will be described using java
        piece of code.
        Such code is now encapsulated in XML document description OR
        the file containing the code may be attached to the XML
        description.
     This may provide a practical manner to describe complex algorithms...
     It is also possible to encapsulated anything else the end-user want
to.


     The following example illustrates the java code encapsulation
     (notice the special character &lt; as "<" :

     <CODE-PIECE xml:space="preserve">
        public class HelloWorld {

            public static void main (String[] args) {
                for (int i=0; i &lt; 10; i++) {
                    System.out.println("HelloWorld - Number : " + i);
                }
            }
        }
     </CODE-PIECE>

<!--  -->
```

```
<!-- ============================================================
        Date:           09/04/2001
        Author:         Herve Verjus
        Revised:

        File: condition.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe CONDITION elements.
     ============================================================ -->


<!-- The root elements of the XML document associated with this DTD is
     the CONDITION.
-->

<!-- It is a minimum set of grammar description limited to binary
    operators. If required, section will be extended later.
-->
<!ELEMENT CONDITION     ( OPERAND, OPERATOR, OPERAND )>
<!-- If the condition is negative condition -->
<!ATTLIST CONDITION     not             (yes | no)  "no"    #IMPLIED>

<!ELEMENT OPERAND       EMPTY>
<!ATTLIST OPERAND       value           CDATA               #REQUIRED>
<!-- If the operande is a variable (i.e. and ID) or not -->
<!ATTLIST OPERAND       variable        (yes | no)  "no"    #IMPLIED>

<!ELEMENT OPERATOR      EMPTY>
<!ATTLIST OPERATOR      type            %OperatorTypeList;  #REQUIRED>

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % OperatorTypeList "('SUB' | 'ADD' | 'MULT' | 'DIV' |
            'CONTAINS' | 'INF' | 'SUP' | 'EQUALS' )">
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: models.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe MODELS elements. Models
            are un set of MODEL elements.


     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the MODELS tag which is a set of MODEL.
-->
<!ELEMENT MODELS        (MODELS-DESC?, MODEL*)>
<!ELEMENT MODELS-DESC   #PCDATA>
<!ATTLIST MODELS        comment                 CDATA       #IMPLIED>
<!ENTITY % MODEL        SYSTEM                  "model.dtd">



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: model.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe MODEL elements.
            MODEL (of tool) is a similarly concept as the concept
            of car's model. A "real" tool is a MODEL's instance...

     ============================================================ -->


<!-- DTD for the models of components -->

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % MessageServerList "('JMS')">
<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % RemoteMethodServerList "('RMI' | 'CORBA' | 'DCOM' )">
<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % LanguageList "('java' | 'C' | 'C++' | 'Ada' | 'SmalTalk')">


<!-- The root elements of the XML document associated with this DTD is
     the MODEL tag.
-->

<!-- Model allowing to describe 'really' components 'off-the-shelves' ! -->
<!ELEMENT MODEL                 (MODEL-DESC?, COMPONENT-INTERFACE*,
CONTEXT*,
```

```
            (IMPORT-FORMAT* | "all"), (EXPORT-FORMAT* | "all"), MODEL-INIT
)>
<!ATTLIST MODEL      modelID             ID
#REQUIRED>
<!ATTLIST MODEL      client-serveur      ("yes" | "no")             "no">
<!ATTLIST MODEL      client-serveur-part ("client" | "server")
#IMPLIED>
<!ATTLIST MODEL      portable            ("yes" | "no")             "no">
<!ATTLIST MODEL      daemon              ("yes" | "no")             "no">
<!ATTLIST MODEL      persistent-state    ("yes" | "no")
#IMPLIED>
<!ATTLIST MODEL      modify-FS           ("yes" | "no")
#IMPLIED>
<!ATTLIST MODEL      generates-events    ("yes" | "no")             "no">
<!ATTLIST MODEL      user-interaction    ("yes" | "no")
#IMPLIED>
<!ATTLIST MODEL      version             CDATA
#IMPLIED>

<!ELEMENT MODEL-DESC            (#PCDATA)>

<!ELEMENT COMPONENT-INTERFACE (UnP-SERVICE-INTERFACE | WnP-SERVICE-
INTERFACE)>

<!-- UnP stands for Use-and-Play: all services that are available 'as is'
-->
<!-- may be directly invoked without wrapping... -->
<!-- WnP stands for Wrapp-and-Play: all services that have to be wrapped  -
->
<!-- before they are available for using  -->
<!ELEMENT UnP-SERVICE-INTERFACE
                          (UnP-OPERATION-INTERFACE | UnP-MESSAGE-
INTERFACE)>

<!ELEMENT UnP-OPERATION-INTERFACE
         (OPERATIONS, InitI-OPERATIONS?, InitC-OPERATIONS?, Conf-
OPERATIONS?)>

<!ATTLIST UnP-OPERATION-INTERFACE
            type                    CDATA %RemoteMethodServerList;
#REQUIRED>
<!ATTLIST UnP-OPERATION-INTERFACE
            name                    CDATA
#REQUIRED>


<!-- === Begin of OPERATIONS description === -->
<!ELEMENT OPERATIONS (OPERATION*)>

<!-- Methods that have been invoked to initialize the interface -->
<!ELEMENT InitI-OPERATIONS (OPERATION | OPERATION-ID)*>

<!-- Methods that have been invoked to inilialize the component -->
<!ELEMENT InitC-OPERATIONS (OPERATION | OPERATION-ID)*>

<!-- Methods that have been invoked to configure the interface -->
<!ELEMENT Conf-OPERATIONS (OPERATION | OPERATION-ID)*>

<!-- OPERATION-ID corresponds to an existing OPERATION's
operationID declaration -->
<!ELEMENT OPERATION-ID          (#PCDATA)>
<!-- === End of OPERATIONS description === -->

<!-- === Begin of UnP-MESSAGE-INTERFACE description === -->
```

```
<!ELEMENT UnP-MESSAGE-INTERFACE
                (MESSAGES, InitI-MESSAGES?, InitC-MESSAGES?, Conf-
MESSAGES?)>

<!ATTLIST UnP-MESSAGE-INTERFACE type %MessageServerList; #REQUIRED>
<!-- === End of UnP-MESSAGE-INTERFACE description === -->


<!-- === Begin of MESSAGES description === -->
<!ELEMENT MESSAGES (MESSAGE*)>

<!-- Messages that have been sent to initialize the interface -->
<!ELEMENT InitI-MESSAGES (MESSAGE | MESSAGE-ID)*>

<!-- Messages that have been sent to inilialize the component -->
<!ELEMENT InitC-MESSAGES (MESSAGE | MESSAGE-ID)*>

<!-- Messages that have been sent to configure the interface -->
<!ELEMENT Conf-MESSAGES (MESSAGE | MESSAGE-ID)*>
<!--  -->

<!-- MESSAGE-ID corresponds to an existing MESSAGE's messageID declaration
-->
<!ELEMENT MESSAGE-ID            (#PCDATA)>
<!-- === End of OPERATIONS description === -->


<!-- === Begin of WnP-SERVICE-INTERFACE description === -->
<!ELEMENT WnP-SERVICE-INTERFACE
    (WnP-OPERATION-INTERFACE | USER-INTERFACE | COMMAND-INTERFACE |
LIBRARY)>

<!ELEMENT WnP-OPERATION-INTERFACE
        (OPERATIONS, InitI-OPERATIONS?, InitC-OPERATIONS?, Conf-
OPERATIONS?)>

<!ATTLIST WnP-OPERATION-INTERFACE type %LanguageList; >
<!--  -->

<!ELEMENT USER-INTERFACE
        (NATIVE-SERVICES, InitI-NATIVE-SERVICES?, InitC-NATIVE-SERVICES?,
                                                  Conf-NATIVE-SERVICES?)>
<!ATTLIST USER-INTERFACE level CDATA>

<!-- === Begin of NATIVE-SERVICES description === -->
<!ELEMENT NATIVE-SERVICES (NATIVE-SERVICE*)>

<!-- Services that have been invoked to initialize the interface -->
<!ELEMENT InitI-NATIVE-SERVICES (NATIVE-SERVICE | NATIVE-SERVICE-ID)*>

<!-- Services that have been invoked to initialize the component -->
<!ELEMENT InitC-NATIVE-SERVICES (NATIVE-SERVICE | NATIVE-SERVICE-ID)*>

<!-- Services that have been invoked to configure the interface -->
<!ELEMENT Conf-NATIVE-SERVICES (NATIVE-SERVICE | NATIVE-SERVICE-ID)*>
<!--  -->

<!-- NATIVE-SERVICE-ID corresponds to an existing NATIVE-SERVICE's
nativeServiceID declaration -->
<!ELEMENT NATIVE-SERVICE-ID     (#PCDATA)>
<!-- === End of NATIVE-SERVICES description === -->

<!--  -->
<!ELEMENT COMMAND-INTERFACE (COMMANDS, InitC-COMMANDS)>
```

```
<!-- === Begin of COMMANDS description === -->
<!ELEMENT COMMANDS (COMMAND*)>

<!-- Commands that have been invoked to inilialize the component -->
<!ELEMENT InitC-COMMANDS (COMMAND | COMMAND-ID)*>

<!-- COMMAND-ID corresponds to an existing COMMAND's commandID declaration
-->
<!ELEMENT COMMAND-ID            (#PCDATA)>
<!-- === End of COMMANDS description === -->


<!-- Library declaration -->
<!ENTITY % LIBRARY          SYSTEM          "library.dtd">


<!--  -->
<!ELEMENT SERVICE (OPERATION | MESSAGE | COMMAND | NATIVE-SERVICE)>

<!ELEMENT SERVICE-ID
        (OPERATION-ID | MESSAGE-ID | COMMAND-ID | NATIVE-SERVICE-ID)>

<!-- Where services DTD are located -->
<!ENTITY % OPERATION        SYSTEM          "operation.dtd">

<!ENTITY % MESSAGE          SYSTEM          "message.dtd">

<!ENTITY % COMMAND          SYSTEM          "command.dtd">

<!ENTITY % NATIVE-SERVICE   SYSTEM          "native-service.dtd">

<!-- Context describing the platform, binaries and path corresponding to
     that platform required by the component (model)
-->
<!ELEMENT CONTEXT (PLATFORM, BINARY*, PATH*)>
<!ATTLIST CONTEXT           contextID       CDATA           #REQUIRED>

<!ENTITY % PLATFORM         SYSTEM          "platform.dtd">

<!ELEMENT BINARY            (#PCDATA)>
<!ATTLIST BINARY            path            CDATA           #IMPLIED>
<!ATTLIST BINARY            init           ("yes" | "no")  "no">

<!ELEMENT PATH              (#PCDATA)>


<!-- Formats that are supported as importation (resp. exportation)
by the component model -->
<!ELEMENT IMPORT-FORMAT     (#PCDATA)>

<!ELEMENT EXPORT-FORMAT     (#PCDATA)>

<!-- Services that are required to initialize the component -->
<!ELEMENT MODEL-INIT        (SERVICE-ID*)>
<!ATTLIST MODEL-INIT        comment         CDATA           #IMPLIED>
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: library.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe LIBRARY elements.
            A library i a set a operations grouped together
            in a sigle file (a DLL, a .so, etc.)

     ============================================================ -->

<!ELEMENT LIBRARY       (OPERATION*)>
<!ATTLIST LIBRARY       libraryID           ID                  #REQUIRED>
<!ATTLIST LIBRARY       comment             CDATA               #IMPLIED>
<!ATTLIST LIBRARY       type                %LibraryTypeList;   #IMPLIED>
<!ATTLIST LIBRARY       fileName            CDATA               #REQUIRED>
<!ATTLIST LIBRARY       path                CDATA               #REQUIRED>

<!ENTITY % OPERATION    SYSTEM              "operation.dtd">

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % LibraryTypeList "('DLL' | 'OCX' | '.h' | '.c' |'.so')">
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: operation.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe OPERATION elements.



     ============================================================ -->

<!--  -->
<!ELEMENT OPERATION    (NAME, PARAMETER*)>
<!ATTLIST OPERATION    operationID          ID          #REQUIRED>
<!ATTLIST OPERATION    comment              CDATA       #IMPLIED>
<!ATTLIST OPERATION    transactional        (yes | no)  "no">
<!-- undo must be a valid operationID : it is the operation undoing what
        this operation has done.  -->
<!ATTLIST OPERATION    undoID               IDREF       #IMPLIED>
<!ATTLIST OPERATION    returnType           CDATA       "void">

<!ELEMENT PARAMETER     EMPTY>
<!ATTLIST PARAMETER     type                 CDATA       #REQUIRED>
<!ATTLIST PARAMETER     name                 CDATA       #REQUIRED>
<!ATTLIST PARAMETER     inOut                CDATA       #IMPLIED>

<!ENTITY % NAME         SYSTEM               "common.dtd">


<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: message.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe MESSAGE elements.



     ============================================================ -->

<!--  -->
<!ELEMENT MESSAGE   (NAME, FIELD*, PAYLOAD?)> <!-- The payload is not
managed yet -->
<!ATTLIST MESSAGE   messageID           ID                  #REQUIRED>
<!ATTLIST MESSAGE   comment             CDATA               #IMPLIED>
<!ATTLIST MESSAGE   transactional       ("yes" | "no")      "no">
<!ATTLIST MESSAGE   undoID              IDREF               #IMPLIED>
<!-- undo must be a valid messageID : it is the message undoing what
        this message has done.  -->

<!ELEMENT FIELD     EMPTY>
```

```
<!ATTLIST PARAMETER name                CDATA               #REQUIRED>
<!ATTLIST PARAMETER value               CDATA               #REQUIRED>

<!ELEMENT PAYLOAD   (#PCDATA)>

<!ENTITY % NAME     SYSTEM              "common.dtd">


<!-- ======================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: command.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe COMMAND elements.



     ====================================================== -->

<!--  -->
<!ELEMENT COMMAND       (NAME, ARGUMENT*)>
<!ATTLIST COMMAND       commandID           ID                  #REQUIRED>
<!ATTLIST COMMAND       comment             CDATA               #IMPLIED>
<!ATTLIST COMMAND       transactional       ("yes" | "no")      "no">
<!-- undo must be a valid commandID : it is the command undoing what
        this command has done.  -->
<!ATTLIST COMMAND       undoID              IDREF               #IMPLIED>

<!ELEMENT ARGUMENT      (#PCDATA | COMMAND | COMMAND-ID)>
<!ATTLIST ARGUMENT      required            ("yes" | "no")      "no">
<!ATTLIST ARGUMENT      type                %ArgumentTypeList;  #IMPLIED>

<!-- must be a valid commandID ! -->
<!ELEMENT COMMAND-ID    (#PCDATA)>

<!ENTITY % NAME         SYSTEM              "common.dtd">

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % ArgumentTypeList "('option' | 'path' | 'file' | 'command' |
'name')">



<!-- ======================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: native-service.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe NATIVE-SERVICE elements.
```

```
     ============================================================ -->

<!ELEMENT NATIVE-SERVICE        EMPTY>
<!ATTLIST NATIVE-SERVICE        nativeServiceID         ID
#REQUIRED>
<!ATTLIST NATIVE-SERVICE        comment                 CDATA
#IMPLIED>
<!ATTLIST NATIVE-SERVICE        transactional           (yes | no)
"no">
<!-- undo must be a valid native-serviceID : it is the native-service
undoing what
        this native-service has done.  -->
<!ATTLIST NATIVE-SERVICE        undoID                  IDREF
#IMPLIED>
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: platform.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe PLATFORM elements.
            PLATFORM describes characteristics of computer and
            (if it is used in a MODEL), computer requirements for
            the corresponding MODEL.

     ============================================================ -->

<!--  -->
<!ELEMENT PLATFORM    EMPTY>
<!ATTLIST PLATFORM    ram             CDATA                   #IMPLIED>
<!ATTLIST PLATFORM    hd              CDATA                   #IMPLIED>
<!ATTLIST PLATFORM    processorArch   %ProcessorArchList;     #IMPLIED>
<!ATTLIST PLATFORM    os              %OSList;                #IMPLIED>
<!ATTLIST PLATFORM    osVersion       CDATA                   #IMPLIED>

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % ProcessorArchList "('i386' | 'i486' | 'Pentium' | 'PentiumII' |
                          'PentiumIII' | 'SPARC' | 'RiscArm' | 'Athlon')">

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % OSList "('WindowsNT' | 'Windows' | 'Linux' | 'AIX' | 'Solaris' |
                                    'Irix' | 'SCO' | 'MacOS' | 'BeOS')">
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: sites.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe SITES elements.
            a SITE is commonly viewed as a set of SITE that is also
            a set of LOCATION.

     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the SITES tag which is a set of SITE.
-->
<!ELEMENT SITES         (SITES-DESC?, SITE*)>
<!ELEMENT SITES-DESC    #PCDATA>
<!ATTLIST SITES         comment         CDATA           #IMPLIED>

<!ELEMENT SITE          (LOCATION*)>
<!-- siteID is a SITE's identifier. It must be unique for a federation -->
<!ATTLIST SITE          siteID          ID              #REQUIRED>
<!ATTLIST SITE          comment         CDATA           #IMPLIED>
<!ATTLIST SITE          domain-name     CDATA           #IMPLIED>
<!ENTITY % LOCATION     SYSTEM          "location.dtd">


<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: location.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe LOCATION elements.
            LOCATION is a specific place on a site: it is commonly
            seen as a machine (with some characteristics) where
            the tool will be installed.

     ============================================================ -->

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY  % NetTypeList "('BANYAN' | 'TOKENRING' )">
<!-- The following enumeration list may not be exhaustive -->
<!ENTITY  % NetConnectivityList "('10Base2' | '10BaseT' )">
<!-- The following enumeration list may not be exhaustive -->
<!ENTITY  % NetProtocolList "('TCP' | 'UDP' | 'NetBeui' | 'IPX' )">
<!-- The following enumeration list may not be exhaustive -->
<!ENTITY  % LServerList "('RMI' | 'JMS' | 'Orbix' )">


<!ELEMENT LOCATION  (NET-INTERFACE*, L-ENV?, L-SERVER*)>
<!ATTLIST LOCATION  locationID          ID                      #REQUIRED>
```

```
<!ELEMENT NET-INTERFACE     (NET-ENV?)>
<!ATTLIST NET-INTERFACE     name        CDATA                   #IMPLIED>
<!ATTLIST NET-INTERFACE     IP          CDATA                   #IMPLIED>

<!ELEMENT NET-ENV   (NET-PROTOCOL*)>
<!ATTLIST NET-ENV   net-adress          CDATA                   #IMPLIED>
<!ATTLIST NET-ENV   net-type            %NetTypeList;           #IMPLIED>
<!ATTLIST NET-ENV   net-connectivity    %NetConnectivityList;   #IMPLIED>

<!ELEMENT NET-PROTOCOL      EMPTY>
<!ATTLIST NET-PROTOCOL      name        %NetProtocolList;       #IMPLIED>

<!ELEMENT L-ENV     (PLATFORM, SOFT-APPLI*)>
<!ENTITY  %         PLATFORM            SYSTEM
"platform.dtd">

<!ELEMENT SOFT-APPLI        (PATH*)>
<!ATTLIST SOFT-APPLI        softID      ID                      #REQUIRED>
<!ATTLIST SOFT-APPLI        binary      CDATA                   #IMPLIED>

<!ELEMENT PATH              (#PCDATA)>

<!ELEMENT L-SERVER  (#PCDATA)>
<!ATTLIST L-SERVER  specifier           %LServerSpecifier;      #IMPLIED>
<!ATTLIST L-SERVER  port                CDATA                   #IMPLIED>
<!ATTLIST L-SERVER  name                CADTA                   #IMPLIED>
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: instances.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe INSTANCES elements.
            Instances are un set of INSTANCE elements.

     ============================================================ -->


<!-- The root elements of the XML document associated with this DTD is
     the INSTANCES tag which represents the federation topology at
     start-up.
-->
<!ELEMENT INSTANCES         (TOPOLOGY-DESC?, (INSTANCE*)>
<!ELEMENT INSTANCES-DESC    #PCDATA>
<!ATTLIST INSTANCES         comment     CDATA>
<!ENTITY % INSTANCE         SYSTEM      "instance.dtd">



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: instance.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe INSTANCE elements.
            Instance is a MODEL's instance, a "real" tool with it
            specific characteristic that are not common to all tools
            that respect the MODEL.

     ============================================================ -->

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % ModeList "('ASYNCH' | 'SYNCH' | 'BOTH')">

<!-- The root elements of the XML document associated with this DTD is
     the INSTANCE tag.
-->

<!ELEMENT INSTANCE      (I-CONTEXT?, COMP-PATH*)>
<!ATTLIST INSTANCE      instanceID      ID                  #REQUIRED>
<!-- modelID corresponds to an existing model's modelID -->
<!ATTLIST INSTANCE      modelID         IDREF               #REQUIRED>
<!-- locationID corresponds to an existing location's locationID
declaration -->
<!ATTLIST INSTANCE      locationID      IDREF               #REQUIRED>
<!ATTLIST INSTANCE      amountOfMemory  CDATA               #IMPLIED>
<!ATTLIST INSTANCE      mode            %ModeList;          #IMPLIED>
```

```
<!-- The I-PLATFORM has to refere a MODEL's modelID's contextID -->
<!ELEMENT I-CONTEXT     (#PCDATA)>
<!ATTLIST I-CONTEXT     comment         CDATA                #IMPLIED>

<!-- Location's paths where the instance is currently located -->
<!ELEMENT COMP-PATH     (#PCDATA)>
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: aspects.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe ASPECTS elements.
            It is a set of ASPECT.

     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the ASPECTS tag which is a set of ASPECT.
-->
<!ELEMENT ASPECTS (ASPECTS-DESC?, ASPECT*)>
<!ELEMENT ASPECTS-DESC #PCDATA>
<!ATTLIST ASPECTS comment CDATA>
<!ENTITY % ASPECT SYSTEM "aspect.dtd">


<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: aspect.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe ASPECT elements.
            Aspect is the core of the federation control system.
            It is hidden stuffs that are executed by the federation
            but are invisible for the tool's point of view.

     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the ASPECT.
-->
<!ELEMENT ASPECT (ASPECT-PARTICIPANT*, CODE-BODY)>
<!ATTLIST ASPECT aspectID       ID      #REQUIRED>
<!-- the transactional mode may be on or off...
The system will check if all services involved in this aspect
are transactional too. If not, the transactional mode will be turned off --
>
<!ATTLIST ASPECT transactional ("yes" | "no") "no"  #IMPLIED>
<!ATTLIST ASPECT comment CDATA                      #IMPLIED>
<!-- the synchTrigger must be a operationID defined in the
                    FEDERAL-SERVICES of the CU tag section -->
<!ATTLIST ASPECT synchTrigger   IDREF               #IMPLIED>
<!-- the asynchTrigger must be a messageID defined in the
                    FEDERAL-SERVICES of the CU tag section -->
<!ATTLIST ASPECT asynchTrigger  IDREF               #IMPLIED>

<!ELEMENT ASPECT-PARTICIPANT EMPTY>
<!-- Has to be a reference to an existing ROLE-ID ! -->
<!ATTLIST ASPECT-PARTICIPANT roleID         CDADA   #REQUIRED>
```

```
<!-- Has to be a reference to an existing SERVICE-ROLE-ID
corresponding to ROLE-ID ! -->
<!ATTLIST ASPECT-PARTICIPANT roleServiceId  CDATA   #REQUIRED>

<!--
================================================================================
        The CODE-BODY allows to code mappings using a programming language
        providing control structures, etc.
        One can be encapsulates whatever the end-user want to..such as
        coordination languages, etc.

================================================================================
-->
<!ENTITY % CODE-BODY SYSTEM "common.dtd">
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: roles.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe ROLES elements. Roles
            are un set of ROLE elements.

     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the ROLES tag which is a set of ROLE.
-->
<!ELEMENT ROLES             (ROLES-DESC?, ROLE*)>
<!ATTLIST ROLES             comment         CDATA       #IMPLIED>

<!ELEMENT ROLES-DESC        (#PCDATA)>

<!ENTITY % ROLE             SYSTEM          "role.dtd">


<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: role.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe ROLE elements.
            ROLE element defines what abstract services that all
            tools implementing/playing the role have to provide...
            and other stuffs like prohibitions.
            Such ROLE concept is mainly a suitable behavior for all
            tools playing it !

     ============================================================ -->

<!-- The following enumeration list may not be exhaustive -->
<!ENTITY % SubscriptionTypeList "('PRE' | 'POST' | 'BOTH')">

<!-- The root elements of the XML document associated with this DTD is
     the ROLE tag.
-->

<!ELEMENT ROLE          (DUTIES?, PROHIBITIONS?>
<!ATTLIST ROLE          roleID          ID                      #REQUIRED>
<!ATTLIST ROLE          comment         CDATA                   #IMPLIED>
<!ATTLIST ROLE          cardinality     CDATA                   #IMPLIED>

<!-- The DUTIES the role has to provide -->
<!ELEMENT DUTIES          (ROLE-SERVICES?, ROLE-SUBSCRIPTIONS?)>

<!ELEMENT ROLE-SERVICES   (ROLE-SERVICE*)>
<!ATTLIST ROLE-SERVICES   comment      CDATA                   #IMPLIED>
```

```
<!ELEMENT ROLE-SERVICE    (OPERATION | MESSAGE)*>

<!ENTITY % OPERATION      SYSTEM       "operation.dtd">
<!ENTITY % MESSAGE        SYSTEM       "message.dtd">

<!ELEMENT ROLE-SUBSCRIPTIONS   (FEDERAL-SERVICE-ID* | "all")>
<!ATTLIST ROLE-SUBSCRIPTIONS   comment         CDATA           #IMPLIED>

<!ELEMENT FEDERAL-SERVICE-ID             #PCDATA>

<!-- The PROHIBITIONS the role is under control = restrictions -->
<!ELEMENT PROHIBITIONS    (FEDERAL-SERVICE-SUBS?, FEDERAL-SERVICE-CALLS?)>

<!ELEMENT FEDERAL-SERVICE-SUBS     (FEDERAL-SERVICE-SUB* | "all")>

<!ELEMENT FEDERAL-SERVICE-SUB      (FEDERAL-SERVICE-ID)>
<!ATTLIST FEDERAL-SERVICE-SUB      type         %SubscriptionTypeList;
#IMPLIED>

<!ELEMENT FEDERAL-SERVICE-CALLS    (FEDERAL-SERVICE-ID* | "all")>



<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:        06/04/2001

        File: role_cooperation.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe ROLE-COOPERATION elements.
            That expresses ALL tools implied in A role.

     ============================================================ -->


<!-- The root elements of the XML document associated with this DTD is
     the ROLE-COOPERATION.
-->
<!ELEMENT ROLE-COOPERATION        (ROLE-COOPERATION-DESC?, ROLE-BEHAVIOR*)>

<!ELEMENT ROLE-COOPERATION-DESC   (#PCDATA)>
<!ATTLIST ROLE-COOPERATION        comment       CDATA            #IMPLIED>
<!-- "Default role adopted when a new tool is joining the federation
        has to be a valid ROLE's roleID or "none" ! -->
<!ATTLIST ROLE-COOPERATION        default       CDATA    "none"  #IMPLIED>


<!ELEMENT ROLE-BEHAVIOR (ROLE-PARTICIPANT*, (ROLE-BODY | CODE-BODY) )>
<!-- Has to be a valid ROLE's roleID ! -->
<!ELEMENT ROLE-BEHAVIOR           roleID        IDREF           #REQUIRED>

<!-- Has to be a valid INSTANCE's instanceID ! -->
<!ELEMENT ROLE-PARTICIPANT        (#PCDATA)>


<!ELEMENT ROLE-BODY               ( (SELECT*) | CODE-BODY)>

<!ELEMENT SELECT                  (CONDITIONS?, SERVICE-PERFORMERS)>
```

```
<!-- Has to be a valid ROLE's operationID OR messageID ! -->
<!ATTLIST SELECT                  roleServiceID  IDREF          #REQUIRED>
<!ATTLIST SELECT                  default        (yes | no) "no"
#IMPLIED>

<!ELEMENT CONDITIONS              (CONDITION*)>

<!ENTITY % CONDITION              "condition.dtd">

<!ELEMENT SERVICE-PERFORMERS      (SERVICE-PERFORMER*)>

<!ELEMENT SERVICE-PERFORMER       EMPTY>
<!-- Has to be a valid INSTANCE's instanceID playing this
ROLE = PARTICIPANT ! -->
<!ATTLIST SERVICE-PERFORMER       instanceID     IDREF          #REQUIRED>

<!--
================================================================================
        The CODE-BODY allows to code mappings using a programming language
        providing control structures, etc.
        One can be encapsulates whatever the end-user want to..such as
        coordination languages, etc.

================================================================================
-->
<!ENTITY % CODE-BODY              SYSTEM        "common.dtd">

<!-- The role does not perform any mappings, transformations:
        it does not know the tool's universe of discourse.
-->
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: wrappers.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe WRAPPERS elements. Wrappers
            are un set of WRAPPER elements.

     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the WRAPPERS tag which is a set of WRAPPER.
-->
<!ELEMENT WRAPPERS             (WRAPPERS-DESC?, WRAPPER*)>
<!ATTLIST WRAPPERS             comment          CDATA        #IMPLIED>

<!ELEMENT WRAPPERS-DESC        (#PCDATA)>

<!ENTITY % WRAPPER             SYSTEM          "wrapper.dtd">


<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: wrapper.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe WRAPPER elements.
            Wrapper defines a wrapper of the tool it encapsulates.

        Important:
            wrappers are not required to the federation description
            while ROLE contains relationships to INSTANCE. Adding wrapper
            requires more expertise but may also provide an unambiguous
            federation description.

     ============================================================ -->

<!ELEMENT WRAPPER            (WRAPPER-INTERFACE, WRAPPING-ACT*,
                         MONITORING-ACT*, STUFF-MONITORED*,
RECOMMANDATION*)>
<!ATTLIST WRAPPER            wrapperID           ID              #REQUIRED>
<!-- instanceID corresponds to an existing INSTANCE's instanceID -->
<!ATTLIST WRAPPER            instanceID          IDREF           #REQUIRED>

<!ELEMENT WRAPPER-INTERFACE  (SYNCH-W-INTERFACE?, ASYNCH-W-INTERFACE?)>

<!ELEMENT SYNCH-W-INTERFACE  (OPERATION*)>

<!ENTITY % OPERATION         SYSTEM              "operation.dtd">

<!ELEMENT ASYNCH-W-INTERFACE (MESSAGE*)>
```

```
<!ENTITY % MESSAGE          SYSTEM              "message.dtd">

<!-- The following section is used for describing a wrapping, i.e.
        a WRAPPER-INTERFACE service that wrapps an INSTANCE's service.
-->
<!ELEMENT WRAPPING-ACT      (CODE-BODY)>
<!ATTLIST WRAPPING-ACT      comment             CDATA           #IMPLIED>
<!-- wrapperServiceID must be a service that is been defined in
    the WRAPPER-INTERFACE definition section
-->
<!ATTLIST WRAPPING-ACT      wrapperServiceID    IDREF           #REQUIRED>
<!-- instanceServiceID must be a valid instance's (or by association, its
        corresponding model's) serviceID (operation, message, command,
etc.)
        that has already been defined in a WnP interface.
-->

<!ATTLIST WRAPPING-ACT      instanceServiceID   IDREF           #REQUIRED>

<!--
================================================================================
        The CODE-BODY allows to code mappings using a programming language
        providing control structures, etc.
        One can be encapsulates whatever the end-user want to..such as
        coordination languages, etc.

================================================================================
-->
<!ENTITY % CODE-BODY        SYSTEM              "common.dtd">

<!-- The following section is used for describing the tool's services using
        monitoring
-->
<!ELEMENT MONITORING-ACT    (CODE-BODY)>
<!ATTLIST MONITORING-ACT    comment             CDATA           #IMPLIED>
<!-- cuServiceID must be a CU'service that is been defined in the CU
        definition section. Often use for synchronization local => CU
purpose.
-->
<!ATTLIST MONITORING-ACT    cuServiceID         IDREF           #REQUIRED>
<!-- instanceServiceID must be a valid instance's (or by association, its
        corresponding model's) serviceID (operation, message, command,
etc.)
        that has already been defined in an interface. A monitoring act
        consists in monitoring the instanceServiceID, detecting when thi
        service is used and calls the corresponding cuServiceID ( dogin
some
        mappings if necessary.
        Mappings are done using java language (in our case) that is
        encapsulated in the CODE-BODY tag.
-->
<!ATTLIST MONITORING-ACT    instanceServiceID   IDREF           #REQUIRED>

<!-- All things that cannot be formally described but the user
want to monitor -->
<!ELEMENT STUFF-MONITORED       (MONITORING-RULE*)>
<!ATTLIST STUFF-MONITORED       comment             CDATA
#IMPLIED>

<!-- Informal description ! Only used for helping end-user ! -->
<!ELEMENT MONITORING-RULE       (ON,ACTION)>

<!ELEMENT ON                    (#PCDATA)>
```

```
<!ELEMENT ACTION                (#PCDATA)>

<!-- Recommandation allowing user to adapt the wrapper... -->
<!ELEMENT RECOMMANDATION        (#PCDATA)>
```

```
<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: proxies.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe PROXIES elements. Proxies
            are un set of PROXY elements.

     ============================================================ -->

<!-- The root elements of the XML document associated with this DTD is
     the PROXIES tag which is a set of PROXY.
-->
<!ELEMENT PROXIES               (PROXIES-DESC?, PROXY*)>
<!ATTLIST PROXIES               comment         CDATA        #IMPLIED>

<!ELEMENT PROXIES-DESC          (#PCDATA)>

<!ENTITY % PROXY                SYSTEM          "proxy.dtd">


<!-- ============================================================
        Date:           03/04/2001
        Author:         Herve Verjus
        Revised:

        File: proxy.dtd

        Document Type Definition
        ------------------------

        Description:
            grammar allowing to describe PROXY elements.
            Proxy is the palce where main mappings are performed.
            It a translator between the federation's universe and
            the tool's universe.

     ============================================================ -->

<!ELEMENT PROXY             (RECOMMANDATION*, MAPPINGS?)>
<!ATTLIST PROXY             proxyID             ID
#REQUIRED>
<!-- instanceID corresponds to an existing WRAPPER's instanceID -->
<!ATTLIST PROXY             instanceID          IDREF
#REQUIRED>
<!-- the PROXY's ability to adapt its behavior, i.e. manage a repository
        containing tool artefacts'state and informations.
-->
<!ATTLIST PROXY             selfAdaptative      ("yes" | "no")
#REQUIRED>

<!-- Recommandation allowing the user to adapt the proxy... -->
<!ELEMENT RECOMMANDATION    (#PCDATA)>

<!ELEMENT STUFF-MONITORED   (#PCDATA)>
<!ATTLIST STUFF-MONITORED   comment             CDATA
#IMPLIED>
```

```
<!-- The proxy has to perform mappings between the Federation's Universe
        of discourse and the tool's Universe of discourse
-->
<!ELEMENT MAPPINGS (MAPPING*)>

<!ELEMENT MAPPING           (CODE-BODY, P-PARTICIPANT*)>
<!-- The role that the instanceID's corresponding proxy participating in...
-->
<!ATTLIST MAPPING           roleServiceID       IDREF
#REQUIRED>

<!-- #PCDATA has to refere to the service that has to be invoked
        and its corresponding destination (the tool's wrapper
        or the tool itself).
-->
<!ELEMENT P-PARTICIPANT     (#PCDATA)>
<!ATTLIST P-PARTICIPANT     destination         ("tool" | "wrapper")
#REQUIRED>

<!--
=============================================================================
        The CODE-BODY allows to code mappings using a programming language
        providing control structures, etc.
        One can be encapsulates whatever the end-user want to..such as
        coordination languages, etc.

=============================================================================
-->
<!ENTITY % CODE-BODY        SYSTEM              "common.dtd">
```

7 Description en FTML à partir de fichiers existants

L'ensemble des documents de DTD figurant dans le paragraphe précédent présentent la grammaire permettant de décrire des fédérations sous forme de documents XML. Une des particularité est que la description résultante est monolithique (un seul document XML décrivant la fédération dont les balises racines et de fin sont <FEDERATION> ... </FEDERATION>). Cependant, les documents de grammaire sont eux modulaires. Permettant, par exemple, de décrire un seul modèle de composant (balise <MODEL>) sans avoir l'obligation de considérer l'ensemble des models (balise <MODELS>).

Aussi, il nous semble pratique de pouvoir décrire des modèles (par exemple) indépendamment les uns des autres et dans des documents XML différents et de pouvoir les regrouper ensuite…Comme il nous apparaît comme intéressant de généraliser cette démarche à l'ensemble des concepts décrits et même de la fédération dans son ensemble (balise <FEDERATION>).

Les fichiers suivants sont des exemples (des patrons) permettant de rendre modulaire la description de fédérations. Ces exemples correspondent au concept de modèles d'outils (et à l'ensemble des modèles d'outils) mais le principe reste le même pour les rôles et les ensembles de rôles, et de façon générale, à l'ensemble des éléments constituants une description de fédération d'outils en FTML (dont certains éléments sont présentés dans le cinquième chapitre).

Pour en savoir plus…

Pour en savoir plus sur la façon de construire une description XML à partir de documents de description existants, vous pouvez consulter le chapitre 9 de [Harold 2000]. Les avantages de rendre modulaire une description de fédérations sont principalement les suivants :

- modularité ;
- réutilisation simplifiée ;
- évolution et changements qui n'affectent qu'un document (module) de la description.

Cela nous permet, entre autre, de faire évoluer la description sans avoir l'obligation de tout redéfinir ce qui nous semble être un avantage important à considérer. Le pendant, d'une telle description sous forme de documents-modules est que le nombre de ces derniers risque d'être important. Il sera dans ce cas intéressant de prévoir une arborescence contenant l'ensemble des fichiers de description; cette arborescente pourrait respecter les différents concepts par exemple.

```
<!-- ====================================================================
     FILE: all_model.dtd

      All lines of this file have to respect the following syntax :

      <!ENTITY model1 SYSTEM "path/model1.xml">

      where:
       - model1 is the name of the model you want to add
       - path is the path of the file model1.xml
       - model1.xml is the description of the model1 model
          (this file is based on the model.dtd DTD file

     ====================================================================

      This example illustrates such a file:

<!ENTITY rcs_for_win SYSTEM "model/rcs_for_win.xml">
<!ENTITY cvs SYSTEM "cvs.xml">

-->
```

```
<!-- ====================================================================
     FILE: all_model.xml



      <?xml version="1.0" standalone="no"?>
      <!DOCTYPE MODELS SYSTEM "models.dtd"> [
      <!ENTITY % models SYSTEM "all_model.dtd">
      %models;
      ]

    >

      where:
       - models.dtd is the dtd this xml file is based on
       - models is a symbolic name
       - all_models.dtd is the file containing descriptions of all the
models

       <MODELS>
        <MODELS-DESC>Short example of a components models set</MODELS-DESC>
        &model1;
        &model2;
        ...


    where model1, model2, etc. are model you want to add in your federation
composition

     ====================================================================

      This example illustrates such a file:


<?xml version="1.0" standalone="no"?>
<!DOCTYPE MODELS SYSTEM "models.dtd"> [
    <!ENTITY % models SYSTEM "all_model.dtd">
    %models;
    ]

>
<MODELS>
    <MODELS-DESC>Short example of a components models set</MODELS-DESC>
    &model1;
    &model2;
    ...

-->
```

Annexe B : Exemple de modèle de fédérations d'outils et traces à l'exécution

1 Classes Java pour le prototype exécutant le scénario de transfert de produit

Un sous-ensemble des classes java du prototype sont présentées dans la suite de ce paragraphe. Nous proposons à la fois les classes qui contiennent les données de la fédération définie pour le scénario de transfert de produit et à la fois des classes génériques (c'est-à-dire indépendantes d'une fédération particulière) implémentant les mécanismes de la fondation de contrôle.

Définition des services fédéraux

```
public interface FederalServices {

 public Activity getActivity(String name);
 public void terminateActivity(process.apel.Activity act, String exitstate);
 public boolean terminateActivity(String toto);

}
```

Implémentation des services fédéraux (API de l'univers commun)

```
public class ACU_API implements FederalServices {

 private static ACU_API_InvocationHandler handler;
 private static FederalServices acu;
 private static Class proxyClass;
 private static RightControlServer srv;
 static {
        srv = new RightControlServer();
        srv.start();

        try {
               handler = new ACU_API_InvocationHandler();
               proxyClass = Proxy.getProxyClass(ACU_API.class.getClassLoader(),
new Class[] { FederalServices.class });  // the proxy class
               acu = (FederalServices) proxyClass.getConstructor(new Class[] {
java.lang.reflect.InvocationHandler.class   }).newInstance(new   Object[]   {
handler });   // the proxy instance
               //supervisor                =                  (AspectTrigger)
Proxy.newProxyInstance(AspectTrigger.class.getClassLoader(),new   Class[]   {
AspectTrigger.class },handler);
    }
```

```
    catch (NoSuchMethodException e) {
        System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the ACU_API class.");
    }
    catch (InstantiationException e) {
      System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the ACU_API class.");
    }
    catch (IllegalAccessException e) {
        System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the ACU_API class.");
    }
    catch (InvocationTargetException e) {
        System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the ACU_API class.");
    }
 }

 private ProcessInstance instance;

 //private

 /* Private constructor allowing to create a dynamic proxy
  */
 private ACU_API(boolean toto) {
        //super();
        instance = new ProcessInstance();
        String className = this.getClass().getName();
        //srv = new RightControlServer();
        //srv.start();
        System.out.println("=== The Foundation Control is now started ...");
        handler.setACU_API(this);
        handler.setRightControlSrv(srv);
/*
        try {
              handler = new ACU_API_InvocationHandler(this,srv);
              proxyClass = Proxy.getProxyClass(ACU_API.class.getClassLoader(),
new Class[] { FederalServices.class });  // the proxy class
              acu = (FederalServices) proxyClass.getConstructor(new Class[] {
java.lang.reflect.InvocationHandler.class    }).newInstance(new    Object[]    {
handler });    // the proxy instance
              //supervisor                  =                  (AspectTrigger)
Proxy.newProxyInstance(AspectTrigger.class.getClassLoader(),new    Class[]    {
AspectTrigger.class },handler);
    }
    catch (NoSuchMethodException e) {
        System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }
    catch (InstantiationException e) {
      System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }
    catch (IllegalAccessException e) {
        System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }
    catch (InvocationTargetException e) {
        System.out.println("=== Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }*/
 }

 /* Public constructors have to call private constructor
  * if it is usefull to create a proxy...
  * Otherwise, private constructor has not to be called
  */
 public ACU_API() {
        this(true);
 }
```

```
 public static RightControlServer getFC() {
       return srv;
 }

 public static FederalServices acu() {
       return acu;

 }

 public Activity getActivity(String name) {
       return instance.getActivity(name);
 }

       public void terminateActivity(Activity act, String existate) {
       // the core of the kernel method is here !
       System.out.println("### Federal Service terminate successfully invoked
!");
       //return true;
 }

 public boolean terminateActivity(String toto) {
       // the core of the kernel method is here !
       System.out.println("### Federal Service terminate successfully invoked
!");
       return true;
 }

}
```

Définition des rôles

Rôle WorkspaceManager

```
public interface WMRole_I1 {

 //public void copy(String file, String host);
 public void transfert(Product p);
 public void transfert(String f);

}
```

Rôle ProcessEngine

```
public interface PERole_I {

 // No service has to be provided by the PE

}
```

Définition des outils[71]

RCS

```
public class RCSWrapper extends Wrapper {

 RCSProxy rcsProxy = null;

 public RCSWrapper(String name) {

       super(name);
 }
```

[71] Pour des soucis de simplification dans le cadre de la première version du prototype, le modèle de chaque outil est défini "dans" chacune des façades. Remarque : il existe une seule instance par outil dans ce cas précis.

```
 public RCSWrapper(String name, String host) {
       super(name,host);
 }

 public void proxy(RCSProxy p) {
       this.rcsProxy = p;
 }

 public RCSProxy proxy() {
       return this.rcsProxy;
 }

       public void create(String f) {
       System.out.println("=== The RCS wrapper is calling the RCS tool for
service request ...");
       boolean execute = doCmd(1,"c:\\Program Files\\ComponentSoftware\\CS-
RCS\\System\\csrcs Create "+f);
 }

 public void checkIn(String f) {
       System.out.println("=== The RCS wrapper is calling the RCS tool for
service request ...");
       boolean execute = doCmd(1,"c:\\Program Files\\ComponentSoftware\\CS-
RCS\\System\\csrcs CheckIn "+f);
 }

static void main(java.lang.String[] args){

  // command a executee
       //String cmd = "javac -classpath "+classesOutDir+" -g -d
"+classesOutDir + " "+outFileDir + "*.java";
       RCSWrapper wrapp = new RCSWrapper("RCS");
       Process compil;
       // Si un shell (DOS) est necessaire pour lancer une commande, ex:
commande system,
        // alors utiliser une des 2 lignes ci-dessous et la concatener avec la
commande voulue !
       String myShell = "cmd /c start cmd ";
       String stopOption = "/c ";  // supprime le shell une fois l'execution
de la commande terminee
       String continueOption = "/k "; // garde le shell present apres la fin
de l'execution de la commande
       String cmd = "dir /p/w";
       wrapp.doCmd(3,cmd);
       //System.out.println(exec);

  }
  */
}
```

```
public class RCSProxy extends Proxy {

 public RCSProxy(String name) {
       super(name);
 }

 public RCSProxy(String n, RCSWrapper wp) {
       this(n);
       wrapper(wp);
 }

 public String getName() {
       return getId();
 }

 public void create(String f) {
       ((RCSWrapper)this.wrapper()).create(f);
 }


 public void transfert(Product p) {
```

```
        String name = p.getName();
        byte[] file = p.getBytes();
        //String filePath = new String("c:\\temp\\");
        java.io.File output = new java.io.File(name); //writes the file
        FileOutputStream streamOUTfileOUT;

        try {
               streamOUTfileOUT = new FileOutputStream(output);
               try {
                      streamOUTfileOUT.write(file);
               }

               catch (IOException ee) {
                      System.out.println("ERROR:"+ee.toString());
               }
        }
        catch (FileNotFoundException e) {
               System.out.println("ERROR: file not found !");
        }
        //String host = ((RCSWrapper)this.wrapper()).host();
        //transfert(filePath,name,host);
        System.out.println("nom :"+name);
        ((RCSWrapper)this.wrapper()).checkIn(name);
 }


 public void transfert(String f) {
        System.out.println("===  The  RCS  proxy  is  performing  all  necessary
translations product->file");
        //copy(f, wrapper().getHost());
        System.out.println("===  The  RCS  proxy  is  invoking  the  RCS  wrapper
...");
        ((RCSWrapper)this.wrapper()).checkIn(f);
 }

}
```

Le moteur de processus

```
public class PEWrapper extends Wrapper {

 PEProxy peProxy = null;

 //private static java.lang.reflect.InvocationHandler handler;
 //private static FederalServices wrapper;
 //private static Class proxyClass;

 public PEWrapper(String name) {

        super(name);

 }

 public PEWrapper(String name, String host) {
        super(name,host);
 }

 public void proxy(PEProxy p) {
        this.peProxy = p;
 }

 public PEProxy proxy() {
        return this.peProxy;
 }

 public boolean start() {
        toolServer.register("ProcessEngine");
        System.out.println("=== PE wrapper invokes the ACU ...");
        Activity act= acu.getActivity("Activity_Specifications");
```

```
        //System.out.println(act.getName());
        acu.terminateActivity(act,"exit");
        System.out.println();
        return true;
  }

}
```

```
public class PEProxy extends Proxy {

  public PEProxy(String name) {
        super(name);
  }

  public PEProxy(String n, PEWrapper wp) {
        this(n);
        wrapper(wp);
  }

  public String getName() {
        return getId();
  }

}
```

Attribution des rôles

```
public class WorkspaceManagerRole extends Role implements WMRole_I1,WMRole_I2
{

  /* All component proxies involved in this role must be
   * declared hereafter !
   */

  private RCSProxy rcs;
  private CVSProxy cvs;

  /* End of proxies declaration */


  public WorkspaceManagerRole(String n) {
        super(n);
        RCSWrapper rcs_wp = new RCSWrapper("rcsWrapp");
        this.rcs = new RCSProxy("rcs", rcs_wp);
        CVSWrapper cvs_wp = new CVSWrapper("cvsWrapp");
        this.cvs = new CVSProxy("cvs", cvs_wp);

        //prohibitions.add(this,"terminateActivity(Product p)", "INVOKE");
  }

  /* All interface implemented methods are declared hereafter */
  /* ======================================================== */
  public void copy(String file, String host) {
  }

  public void transfert(Product p) {
        System.out.println("===  The  role  "+this.getName()+"  is  now  invoked
...)");
        System.out.println("===  ...and  is  identifiying  which  tool  has  to  be
invoked...");
        if ( (p.getName()).startsWith("Spec")) {
              System.out.println("=== The role "+this.getName()+" invoked the
CVS proxy ");
              cvs.transfert(p);
        }
        else {
              System.out.println("=== The role "+this.getName()+" invoked the
RCS proxy");
```

```
                rcs.transfert(p);

        }
 }

 public void transfert(String f) {
        System.out.println("=== The role "+this.getName()+" is now invoked");
        System.out.println("===  ...and  is  identifiying  which  tool  has  to  be
invoked...");
        System.out.println("===  The  role  "+this.getName()+"  is  invoking  the
RCS proxy");
        rcs.transfert(f);
 }

 /* End of methods definition */

}
```

```
public class ProcessEngineRole extends Role implements PERole_I {

 /* All component proxies involved in this role must be
  * declared hereafter !
  */

 private PEProxy pe;

 /* End of proxies declaration */


 public ProcessEngineRole(String n) {
        super(n);
        PEWrapper wp = new PEWrapper("peWrapp");
        this.pe = new PEProxy("APEL_PE", wp);
 }

 }
```

Classes qui implémentent la technologie des "proxy" dynamique en java, permettant de capturer les invocations émanant des façades.

```
public class Wrapper {

 public String id;          //format: "fedComponentid##id"
 private String host;
 private String componentModelName;
 private Vector serviceCalls = new Vector();
 //protected FederalServices acu;


 protected java.lang.reflect.InvocationHandler handler;
 protected FederalServices acu;
 protected ToolRegistration toolServer;
 protected Class proxyClass;


 public Wrapper(String wpid) {
        this.id = wpid;
        String className = this.getClass().getName();

                    try {

               handler = new Wrapper_InvocationHandler();
               proxyClass = Proxy.getProxyClass(Wrapper.class.getClassLoader(),
new Class[] { FederalServices.class, ToolRegistration.class });    //       the
proxy class
```

```
              acu = (FederalServices) proxyClass.getConstructor(new Class[] {
java.lang.reflect.InvocationHandler.class    }).newInstance(new    Object[]    {
handler });    // the proxy instance
              toolServer  =  (ToolRegistration)  proxyClass.getConstructor(new
Class[]    {    java.lang.reflect.InvocationHandler.class    }).newInstance(new
Object[] { handler });      // the proxy instance
              //supervisor                  =                  (AspectTrigger)
Proxy.newProxyInstance(AspectTrigger.class.getClassLoader(),new    Class[]    {
AspectTrigger.class },handler);
    }
    catch (NoSuchMethodException e) {
        System.out.println("### Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }
    catch (InstantiationException e) {
      System.out.println("### Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }
    catch (IllegalAccessException e) {
        System.out.println("### Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }
    catch (InvocationTargetException e) {
        System.out.println("### Error "+e.toString()+" within the java proxy
creation of the "+className+" class.");
    }

 }

 public Wrapper(String wpid, String machine) {
        this(wpid);
        this.host = machine;
 }

 public String getId() {
        return this.id;
 }

 public String host() {
        return this.host;
 }

/*      public void setACU(FederalServices cu) {
        this.acu = cu;
 }

 public FederalServices getACU() {
        return this.acu;
 }

*/
 public static boolean doCmd(int mode, String cmd) {

        /* Concerns only Windows/shell commands...
           for unix commands, write another doCmd method !!!
         */
        Process execution;
   final java.io.InputStream err;
   final java.io.InputStream output;
   String myCmd = "";
        // lancement de la commande avec creation d'un thread pour lire le
flux de sortie
        try {
              String myShell = "cmd /c start cmd ";
              String  stopOption  =  "/c  ";   //  supprime  le  shell  une  fois
l'execution de la commande terminee
              String continueOption = "/k "; // garde le shell present apres
la fin de l'execution de la commande
                           switch(mode) {
                                 case 1:      // sans creation de shell
                                       myCmd = cmd;
```

```
                                        break;
                                    case 2:                   // avec shell non-
presistent apres l'execution de la commande
                                        myCmd = "cmd /c start cmd /c "+cmd;
                                      break;
                                    case 3:                   //    avec    shell
presistent apres l'execution de la commande
                                        myCmd = "cmd /c start cmd /k "+cmd;
                                        break;
                                    default:
                                        myCmd = cmd;
                               }
                                        System.out.println("before   exec   de   "+cmd
+mode);
            execution = java.lang.Runtime.getRuntime().exec(myCmd);
            err = execution.getErrorStream();
            output = execution.getInputStream();
            Thread thread = new Thread () {
                                public void run() {
                                    int ch;
                                try {
                                    while ((ch = err.read()) != -1)
                                            System.out.write(ch);
                                } catch (java.io.IOException e) {
                                    }
                                }
                             };
            thread.run();

        } catch (java.io.IOException e) {
            System.out.println(e.toString());
                                return false;
        }
                    // attente de la fin de la commande
       int result = 0;
        try {
            result = execution.waitFor(); // attente de la fin de l'execution
de la cmd
        } catch (java.lang.InterruptedException e) {
            //Compiler.internalError(e.toString());
        }
        if (result != 0) {
            //        Compiler.internalError(out+"\n"+err);
            //error(0, "can't execute command");
                    System.out.println("Can't execute command");
            return false;
        } else {
                    int ch;
                 try {
                while ((ch = err.read()) != -1)
                   System.out.write(ch);
                }
                catch (java.io.IOException e) {
                }


                    System.out.println("ok");

                    return true;
        }
   }

}
```

```
public          class          Wrapper_InvocationHandler          implements
java.lang.reflect.InvocationHandler {


 //private proto.stable.syntax.acu.kernel.ACU_API api;
 private FederalServices api;
```

```
 private RightControlServer srv;

 private String toolID;

 private Vector invocations = new Vector();    // where all invocation Thread
are stored

 public Wrapper_InvocationHandler() {
       super();
       api = ACU_API.acu();
       srv = ACU_API.getFC();
 }

 /*public     Wrapper_InvocationHandler(proto.stable.syntax.acu.kernel.ACU_API
impl, RightControlServer rghtCtrlSrv){
       super();
       api = impl;
       this.rightControlsrv = rghtCtrlSrv;
 }

 */
 public Object invoke(Object proxy, java.lang.reflect.Method method, Object[]
args) throws Throwable {
       System.out.println("===    Federal    Service    "+method.getName()+"
invocation has been caught by the Foundation Control...");
              try {
                     if ( srv.isAuthorized(toolID, method.getName(), "INVOKE")
) {

                            /*if ( method.getName().equals("getActivity") ) {
                                   //System.out.println("===     The     current
invocation is AUTHORIZED !!!");
                                   return
(Activity)(api.getActivity((String)args[0]));
                            }*/
       //                   else {
                                   //System.out.println("===     The     current
invocation is PROHIBITED !!!");
       //                          return null;
       //                   }
                            if ( method.getName().equals("register") ) {
                                   toolID = srv.register((String)args[0]);
                                   System.out.println("===       The       tool
"+(String)args[0]+" has the ID "+toolID);
                                   return null;
                            }
                            else {
                                   Object result = method.invoke(api, args);
                                   return result;
                            }
                     }
                     else
                            return null;
                     //ACU_API_Invocation       apiInvocation       =       new
ACU_API_Invocation(currentInvocationID, api, aspectsBodies);
                     //apiInvocation.start();
                     //this.invocations.addElement(apiInvocation);
 // the new Trigger Thread is now stored for futur deletion
                     //apiInvocation.existingAspects(aspectsDecl); //  send  all
the aspects declarations
                     //Object  result  =  apiInvocation.invoke(method,  args);
                     // transfert kernel method is now called...
                     //apiInvocation = null; // the trigger invocation thread
is assigned with the null value for garbaging
                     //return null;
              }
              catch (UndeclaredThrowableException e) {
                     System.out.println("*** ERROR in the invoke method");
                     return null;
              }
```

```
  }

}
```

Le serveur de contrôle des droits : implémentation des interdictions

```
public class RightControlServer extends java.lang.Thread  {

  private Vector tools = new Vector();
  private Vector prohibitions = new Vector();
  private boolean ok = false;

  public RightControlServer() {}

  public void run() {
        System.out.println("=== The Right Control Server is started ");
        System.out.println("=== Federation model is being loaded ...");
        init();

        //while (true) { }
  }

/*      public void addProhibition(Role r, String sf, String m) {
        Prohibition p = new Prohibition(r, sf, m);
        this.prohibitions.addElement(sf);
  }

  public boolean checkInvocation(Role r, String sf) {
        //Prohibition
        boolean found = false;
        int i=0;
        while (!found && i<prohibitions.size() ) {
              if ( ((Prohibition)prohibitions.elementAt(i)).getRole() == r &&

  ((Prohibition)prohibitions.elementAt(i)).getFederalService().equals(sf) &&

  ((Prohibition)prohibitions.elementAt(i)).getMode().equals("INVOKE") )
                                    found = true;
              else
                    i++;
        }
        return found;
  }

  */
  public void init() {
        Tool pe = new Tool("ProcessEngine");
        pe.setId("tool-001");
        Tool rcs = new Tool("RCS");
        rcs.setId("tool-002");
        Tool cvs = new Tool("CVS");
        cvs.setId("tool-003");
        tools.addElement(pe);
        tools.addElement(rcs);
        tools.addElement(cvs);
        Prohibition p1= new Prohibition("tool-001", "getActivity", "OBSERVE");
        prohibitions.addElement(p1);
        Prohibition   p2=   new   Prohibition("tool-001",   "terminateActivity",
"OBSERVE");
        prohibitions.addElement(p2);

  }

  public String getId(String tool) {
        boolean found = false;
        int i = 0;
        Tool currentTool;
        while (i < tools.size() && found == false) {
              currentTool = (Tool)tools.elementAt(i);
              if ( (currentTool.getName()).equals(tool) ) {
                    found = true;
```

```
                        return currentTool.getId();
                }
                else
                        i++;
        }
        return "unknown";
 }

 public String register(String tool) {
        boolean found = false;
        int i = 0;
        Tool currentTool;
        String result = "unknown";
        while (i < tools.size() && found == false) {
                currentTool = (Tool)tools.elementAt(i);
                if ( (currentTool.getName()).equals(tool) ) {
                        found = true;
                        currentTool.registration(true);
                        System.out.println("=== Tool "+tool+" is now registered
to the federation");
                        result = currentTool.getId();
                }
                else
                        i++;

        }
        return result;
 }


 public boolean isAuthorized(String tool, String sf, String mode) {
 //     String id = register(tool);
        boolean authorized = true;
 /*     if (id.equals("unknown")) {
                System.out.println("=== The tool "+tool+" is not defined as a
federation member in the federation model");
                System.out.println("=== ");

                authorized=false;
        }*/
        boolean found = false;
        int i = 0;
        Prohibition currentProhibition;
        while (i < prohibitions.size() && found == false) {
                currentProhibition = (Prohibition)prohibitions.elementAt(i);
                if      (      (currentProhibition.getTool()).equals(tool)      &&
(currentProhibition.getFederalService()).equals(sf)                             &&
(currentProhibition.getMode()).equals(mode)) {
                        found = true;
                        //System.out.println("=== Tool "+tool+" is now registered
to the federation");
                        authorized = false;
                }
                else
                        i++;

        }
        if ( authorized ) {
                System.out.println("===  The  current  invocation  is  AUTHORIZED
!!!");
        }
        else {
                System.out.println("===  The  current  invocation  is  PROHIBITED
!!!");
        }
        return authorized;
 }


}
```

Déclaration des aspects

```
public interface Aspects {

 public void terminateActivity_A1(process.apel.Activity act, String existate);
 public void terminateActivity_A2(String existate);

}
```

```
public class AspectsDeclaration {

 private Vector aspects = new Vector();
 //private Aspects aspects = new Aspects();

 public AspectsDeclaration() {


        // All aspects triggers are declared hereafter
        //  All  arguments  class  names  have  to  be  fully-qualified  (i.e.
java.lang.String...)
        Method m1 = new Method("terminateActivity","void");
        //m1.addArg("Activity","act");
        m1.addArg("process.apel.Activity", "act");
        m1.addArg("java.lang.String", "exitstate");
        Aspect a1 = new Aspect("terminateActivity_A1");
        a1.trigger(m1);
        aspects.addElement(a1);
        System.out.println("display "+m1.displayOptionList());

        Method m2 = new Method("terminateActivity","boolean");
        //m2.addArg("Activity","act");
        m2.addArg("java.lang.String", "existate");
        Aspect a2 = new Aspect("terminateActivity_A2");
        a2.trigger(m2);
        aspects.addElement(a2);



        /* End of aspects triggers declaration */
        // etc....etc...
 }
 public Vector aspects() {
        return this.aspects;
 }

}
```

Implication des rôles dans l'exécution des aspects

```
public class AspectsBodies extends java.lang.Thread implements Aspects {

 private String aspectIDtoExec;
 private ACU_API_Invocation apiInvocation;

 /* All the roles involved in aspects must be declared hereafter
  */
 private           WorkspaceManagerRole           wm           =           new
WorkspaceManagerRole("WorkspaceManager");            //   the   VersionManager
role
 // ...
 // ...

 public AspectsBodies() {
        super();
 }

 public AspectsBodies(ACU_API_Invocation t) {
        this();
```

```
        this.apiInvocation = t;
  }

  public AspectsBodies(ACU_API_Invocation parent, String n) {
        this(parent);
        this.aspectIDtoExec = n;
  }

  public String id() {
        return this.aspectIDtoExec;
  }

  public void start() {
        System.out.println("===   The   aspect   THREAD   "+   aspectIDtoExec+"
triggered by "+apiInvocation.id()+" is NOW invoked");
  }

  public void terminateActivity_A1(process.apel.Activity act, String existate)
{
//      public boolean terminate_A1(String existate) {
        System.out.println(">>> In aspect body terminateActivity_A1 <<<");
        System.out.println(">>> The ASPECT is invoking the role ...");
        //return true; //pe.terminate(act,existate);               //
transfert kernel method is now called...
        Product[] products = act.getProducts();
        for (int i=0; i<products.length; i++) {
              if (products[i] != null) {
                    System.out.println("ds products");
                    wm.transfert(products[i]);                    //       put
method called of the role WM
              }
        }
        System.out.println(">>>   The   aspect   is   sending   to   the   Control
Foundation the normal termination signal");
        apiInvocation.terminated(this);
  }

  public void terminateActivity_A2(String existate) {
//            Product[] products = act.getProducts();
//            for (int i=0; i<products.length; i++)
//                  wm.transfert(products[i]);                    //       put
method called of the role WM
        System.out.println(">>> In aspect body terminateActivity_A2 <<<");
        //return true; //pe.terminate(act,existate);                     //
transfert kernel method is now called...
        System.out.println(">>> The ASPECT is invoking the role ...");
        wm.transfert("c:\\temp\\doc_de_specs.doc");
        System.out.println(">>>   The   aspect   is   sending   to   the   Control
Foundation the normal termination signal");
        apiInvocation.terminated(this);

  }

}
```

Prise en charge d'une invocation d'un service fédéral par la fondation de contrôle (par mécanisme d'introspection), gestion du contrôle

```
public class ACU_API_Invocation extends java.lang.Thread {

  private String ACU_API_InvocationID;
  //private java.lang.reflect.Method method;
  //private java.lang.Object[] args;


  private AspectsDeclaration aspectsDecl;// where aspects are declared
  private federation.acu.kernel.ACU_API api;

  private Vector aspectsTriggered = new Vector();
```

```
 private boolean execAPI_Method = false;
 private int numbOfAspectsInvoked = 0;  // The number of aspects triggered by
the same method invocation
 private AspectsBodies aspectsBodies;          //  where  aspects  bodies  are
defined

 public ACU_API_Invocation() {}

 public  ACU_API_Invocation(String  id,  federation.acu.kernel.ACU_API  kernel,
AspectsBodies bodies) {
       //this.method = m;
       //this.args = argS;
       this.ACU_API_InvocationID = id;
       this.api = kernel;
       this.aspectsBodies = bodies;
 }

 public String id() {
       return this.ACU_API_InvocationID;
 }

 public void existingAspects(AspectsDeclaration decl) {
       this.aspectsDecl = decl;
 }

 public  java.lang.Object  invoke(java.lang.reflect.Method  method,  Object[]
args) {
       try {

             ACU_APIPerformer apiExec = new ACU_APIPerformer(this, api);
             apiExec.start();
             //AspectSynchronizer       aspectSynchronizer       =       new
AspectSynchronizer(kernelExec,currentInvocationID);
             //counter.start();
                   /* End of the STEP 1 */
                   /*  Second  STEP  :  all  aspects  whose  triggers  match  the
method invocation are invoked */
                   for (int i=0; i<aspectsDecl.aspects().size(); i++) {
                         //System.out.println("Match        result        :"
+match(method,  args,  ((Aspect)(aspectsDecl.aspects().elementAt(i))).triggers()
) );
                        if          (          match(method,          args,
((Aspect)(aspectsDecl.aspects().elementAt(i))).triggers() ) ) {
                                    java.lang.reflect.Method[]   methodsForAspect
= aspectsBodies.getClass().getMethods(); // collects all aspects bodies
                                    java.lang.reflect.Method   methodOnAspect   =
method;
                                    for   (int   j=0;   j<methodsForAspect.length;
j++) {
                                          if                                    (
methodsForAspect[j].getName().equals(
((Aspect)aspectsDecl.aspects().elementAt(i)).getName() ) )
                                                methodOnAspect                =
methodsForAspect[j];
                                    }
                                    System.out.println("                       =>
"+methodOnAspect.getName()+"  "+args.length);
                                    numbOfAspectsInvoked++;   //  the  number  of
triggered aspects is incremented
                                    AspectsBodies     currentAspect     =     new
AspectsBodies(this,((Aspect)aspectsDecl.aspects().elementAt(i)).getName());
                                    currentAspect.start();

 this.aspectsTriggered.addElement(currentAspect);    // add the aspect thread
to the list of triggered aspects
                                    try {
                                          System.out.println("=== The aspect "+
((Aspect)aspectsDecl.aspects().elementAt(i)).getName()+" is TERMINATED");
                                          Object            result            =
methodOnAspect.invoke(currentAspect, args);
                                    }
```

```
                                    catch (java.lang.IllegalArgumentException e)
{
                                          System.out.println("***  ERROR:  wrong
number of arguments in the aspect invocation !!!");
                                          System.out.println("*** The number of
aspect arguments and those of the method triggering the aspect does not match
===");
                                          System.out.println("***         Aspect
"+methodOnAspect.getName()+" has been canceled...");
                                          remove(currentAspect);
                                          numbOfAspectsInvoked--;

                                    }
                                    catch (java.lang.IllegalAccessException e) {
                                          System.out.println("***  ERROR  in  the
invoke aspect thread method "+e.toString()+"");
                                          System.out.println("***         Aspect
"+methodOnAspect.getName()+" has been canceled...");
                                          remove(currentAspect);
                                          numbOfAspectsInvoked--;

                                    }
                                    catch
(java.lang.reflect.InvocationTargetException e) {
                                          System.out.println("***  ERROR  in  the
invoke aspect thread method "+e.toString()+"");
                                          System.out.println("***         Aspect
"+methodOnAspect.getName()+" has been canceled...");
                                          remove(currentAspect);
                                          numbOfAspectsInvoked--;

                                    }
                              }
                        }
                        System.out.println("=== The number of aspects THREAD for
the     trigger     invocation     "+this.ACU_API_InvocationID+"     is     :
"+numbOfAspectsInvoked);
                        while (!this.execAPI_Method && numbOfAspectsInvoked != 0)
{
                              //  If  there  is  no  aspect'trigger  matching  the
kernel method invocation
                              // the kernel method has to be directly invoked.
                              // Else, the kernel method has not to be invoked
                              // until one aspect thread is finished
                        }
                        System.out.println("===  At  least,  one  aspect  thread  is
terminated");
                        System.out.println("=== Before the Federal Service thread
invocation");
                        //kernelExec.doIt();
                  try {
                              Object result = apiExec.invoke(method, args);
                  // // transfert kernel method is now called...
                              while (!allAspectsTerminated() ) {
                                    //System.out.println("=== "+allAsp
                                    //  while  all  aspects  tiggered  are  not
terminated, the result
                                    // of the initial call is not sent
                              }
                              //  ...  then,  the  result  is  sent  (i.e.  no  more
aspects thread are declared
                              apiExec = null;     // the KernelMethodPerformer is
now assigned to null for garbaging
                              return result;
                  }
                        catch  (java.lang.reflect.UndeclaredThrowableException  e)
{
                              System.out.println("***   ERROR   in   the   invoke
ACU_API_Invocation method "+e.toString());
                              return null;
                        }
```

```
              /*    catch (java.lang.IllegalAccessException e) {
                           System.out.println("***   ERROR   in   the   invoke
ACU_API_Invocation method "+e.toString());
                           return null;
                     }
                     catch (java.lang.reflect.InvocationTargetException e) {
                           System.out.println("***   ERROR   in   the   invoke
ACU_API_Invocation method "+e.toString());
                           return null;
                     }*/
              }

       catch (java.lang.reflect.UndeclaredThrowableException e) {
              System.out.println("*** ERROR in the invoke method");
              return null;
       }

 }

 // This method test whether or not the kernel method call is matching
 // one of the trigger of one or more aspects.
 private   boolean   match(java.lang.reflect.Method   method,   Object[]   args,
Trigger[] trigger) {
       boolean matchOK = false;
       int t = 0;
       Trigger current = trigger[t];
       while (t<2 && current != null) {
              //System.out.println("Ds match "+current.getClass().getName());
              if                                                              (
(current.getClass().getName()).equals("federation.aspect.trigger.Method") ) {
                     System.out.println("===  The  aspect  trigger  currently
tested IS A method ===");
                     // when the arguments are method parameters
                     if ((method.getName()).equals(current.getName()) ) {
                           boolean diff = false;
                           if                                                 (
((method.getReturnType()).getName()).equals(((Method)current).getReturnClass()
) ) {
                                  for (int i=0; i<args.length; i++) {
                                        //System.out.println(" Args class name
= "+args[i].getClass().getName());
                                        //System.out.println("trigger
"+args[i].getClass().getName()+                    "                      --
"+((Parameter)(current.args()[i])).getClassName());
                                        if  ((Parameter)current.args()[i]  ==
null) {
                                              diff=true;
                                        }
                                        else
                                              if             (             !
args[i].getClass().getName().equals(
((Parameter)(current.args()[i])).getClassName()) )
                                                    diff = true;
                                  }
                                  if    ((Parameter)current.args()[args.length]
!= null)
                                              diff=true;
                           }
                           else
                                  diff = true;
                           if (diff == false) {
                                  System.out.println("===  The  Federal  Service
currently invoked and the aspect's trigger currently tested are identical");
                                  matchOK = true;
                           }
                           else {
                                  System.out.println("===  The  Federal  Service
currently invoked and the aspect's trigger currently tested are different");

                           }
                     }
```

```
                }
                t++;
                current = trigger[t];
        }
        System.out.println(matchOK);
        return matchOK;
 }

 public void terminated(AspectsBodies aspectThread) {
        this.execAPI_Method = true;
        remove(aspectThread);
        aspectThread = null;      // the aspect thread reference is fixed to
null for garbaging
 }

 private void remove(AspectsBodies aspectThread) {
        System.out.println("=== Aspect "+aspectThread.id()+" has been removed
from the aspects list !");
        aspectsTriggered.removeElement(aspectThread);
 }

 private boolean allAspectsTerminated() {
        return aspectsTriggered.isEmpty();
 }


}
```

Exécution effective des services fédéraux et gestion du contrôle

```
public class ACU_APIPerformer extends java.lang.Thread {

 private federation.acu.kernel.ACU_API api;
 private ACU_API_Invocation apiInvocation;

 private boolean ok = false;

 public ACU_APIPerformer() {}

 public ACU_APIPerformer(ACU_API_Invocation t, ACU_API p) {
        this.api = p;
        this.apiInvocation = t;
 }

 public Object invoke(java.lang.reflect.Method m, Object[] argS) {
        //while (!this.ok) {
        //}
        try {
                Object result = m.invoke(api, argS);   //   the   kernel   method
invocation is then returned
                System.out.println("===    The    FEDERAL    SERVICE    has    been
successfully executed");
                return result;
        }
        catch (java.lang.IllegalAccessException e) {
                System.out.println("*** ERROR in the invoke kernel method");
                return null;
        }
        catch (java.lang.reflect.InvocationTargetException e) {
                System.out.println("*** ERROR in the invoke kernel method");
                return null;
        }
 }

 public void doIt() {
        this.ok = true;
 }

}
```

2 Modèle en langage FTML

L'exemple ci-dessous concerne un modèle d'une fédération inspirée du scénario de transfert de produit. Le modèle décrit un comportement "dictatorial" avec la description d'un aspect. Comme nous l'avons mentionné dans le cinquième chapitre, les modifications à apporter à ce modèle pour que la fédération évolue vers un comportement "anarchique" sont minimes :

- suppression de la section délimitée par les balises <ASPECTS> ... </ASPECTS> ;
- modification de la définition du rôle *WorkspaceManager* comme illustrée dans la section V.5.3.

```
<?xml version="1.0"?>
<!DOCTYPE FEDERATION SYSTEM "federation.dtd">

<FEDERATION>

  <CU-SET>
   <!-- Description of the ACU API = set of federal services -->
   <CU cuID="scenario1">
    <CU-DESC>A very simple Common Universe</CU-DESC>
    <FEDERAL-SERVICES>
      <OPERATION operationID="terminateActivity" transactional="no"
                 returnType="void">
          <NAME>terminate</NAME>
          <PARAMETER type="RMI_Activity" name="act"></PARAMETER>
          <PARAMETER type="String" name="exitstate"></PARAMETER>
      </OPERATION>
      <!-- have to be completed if necessary -->
      <!-- sf1 -->
      <!-- sf2 -->
      <!-- ... -->
      <!-- sfn -->
    </FEDERAL-SERVICES>
   </CU>
  </CU-SET>

  <ROLES>
    <!-- Description of the WorkspaceManager ROLE -->
   <ROLE roleID="WorkspaceManager">
    <DUTIES>
        <ROLE-SERVICES>
            <OPERATION operationID="productTransfert"
      comment="a product transfert between source activity and a target one"
      transactional="no" returnType="void">
                <NAME>transfert</NAME>
                <PARAMETER type="RMI_Product" name="p">
            </OPERATION>
            <OPERATION operationID="fileTransfert"
      comment="a transfert operation between source activity and a target one"
      transactional="no" returnType="void">
                <NAME>transfert</NAME>
                <PARAMETER type="String" name="f">
            </OPERATION>
        </ROLE-SERVICES>
    </DUTIES>
    <PROHIBITIONS>
        <FEDERAL-SERVICE-CALLS>
            <FEDERAL-SERVICE-ID>terminateActivity</FEDERAL-SERVICE-ID>
        </FEDERAL-SERVICE-CALLS>
    </PROHIBITIONS>
   </ROLE>

   <!-- Other ROLEs may be declared -->
  </ROLES>

  <ASPECTS>
   <!-- Description of the ASPECT expressing ROLE invocation -->
```

```
   <ASPECT aspectID="aspProductTransfert"
    transactional="no" synchTrigger="terminateActivity">
    <ASPECT-PARTICIPANT roleID="WorkspaceManager"
        roleServiceID="productTransfert">
    <CODE-BODY>
        <CODE-PIECE>
            public void terminateActivity(RMI_Activity act, String exitstate)
{
                RMI_Product[] products = act.getProducts();
                for (int i=0; i &lt; products.length; i++)
                    WorkspaceManager.transfert(products[i]);
            }
        </CODE-PIECE>
    </CODE-BODY>
   </ASPECT>
   <!-- Other ASPECTs may be declared -->
  </ASPECTS>


  <SITES comment="Topology of the federation">
    <!-- Description of the SITES = TOPOLOGY of the federation -->
    <SITE siteID="adele" comment="Adele Team site" domain-name="imag.fr">
        <LOCATION locationID="zidane">
            <NET-INTERFACE name="NE001" IP="129.88.103.36">
            <NET-INTERFACE>
            <L-ENV>
                <PLATFORM   ram="128MO"   processorArch="Pentium"   os="Solaris"
version="7">
                </PLATFORM>
            </L-ENV>
        </LOCATION>
        <LOCATION locationID="cantona">
            <NET-INTERFACE name="NE001" IP="189.88.103.39">
            <NET-INTERFACE>
            <L-ENV>
                <PLATFORM   ram="256MO"   processorArch="Pentium"   os="Windows"
version="2000">
                </PLATFORM>
           </L-ENV>
        </LOCATION>
    </SITE>
    <SITE siteID="llp" comment="LLP site" domain-name="univ-savoie.fr">
        <LOCATION locationID="kaboul">
            <NET-INTERFACE name="NE001" IP="193.48.124.79">
            <NET-INTERFACE>
            <L-ENV>
                <PLATFORM   ram="128MO"   processorArch="Pentium"   os="Linux"
version="RedHat 6.2">
                </PLATFORM>
           </L-ENV>
        </LOCATION>
        <LOCATION locationID="verjus">
            <NET-INTERFACE name="NE001" IP="193.48.124.70">
           <NET-INTERFACE>
            <L-ENV>
                <PLATFORM   ram="128MO"   processorArch="Pentium"   os="Windows"
version="2000">
                </PLATFORM>
           </L-ENV>
        </LOCATION>
    </SITE>
  </SITES>

  <MODELS>
    <MODEL modelID="RCSforWindows" client-server="no" portable="no"
     persistent-state="yes" modifyFS="yes" generates-events="no"
     user-interaction="yes">
     <MODEL-DESC>A well known version manager</MODEL-DESC>
     <WnP-SERVICE-INTERFACE>
      <COMMAND-INTERFACE>
         <COMMANDS>
```

```
            <COMMAND commandID="createRCSFile"
                comment="create a new file" transactional="no">
                <NAME>CSRCS create</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
            <COMMAND commandID="checkIn"
                comment="check-in a modified file" transactional="no">
                <NAME>CSRCS CheckIn</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
            <COMMAND commandID="checkOut"
                comment="check-out a working copy" transactional="no">
                <NAME>CSRCS CheckOut</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
            <COMMAND commandID="history"
                comment="trace the revisions file history" transactional="no">
                <NAME>CSRCS History</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
            </COMMAND>
         </COMMANDS>
         <InitC-COMMANDS>
            <COMMAND-ID>createRCSFile</COMMAND-ID>
         </InitC-COMMANDS>
       </COMMAND-INTERFACE>
     </WnP-SERVICE-INTERFACE>
     <CONTEXT contextID="win">
      <PLATFORM processorArch="i486" os="Windows" version="NT 4"></PLATFORM>
      <BINARY path="c:\Program Files\CSRCS\">CSRCS</BINARY>
     </CONTEXT>
    </MODEL>

    <MODEL modelID="ProcessEngine" client-server="no" portable="yes"
     persistent-state="no" modifyFS="no" generates-events="no"
     user-interaction="no" version="2.5.4">
     <MODEL-DESC>The APEL Process Engine</MODEL-DESC>
     <UnP-SERVICE-INTERFACE>
      <UnP-OPERATION-INTERFACE type="RMI" name="PEserver">
        <OPERATIONS>
          <OPERATION operationID="getCharge"
             comment="Get the number of process instances on the process
engine"
             transactional="no" returnType="int">
             <NAME>getCharge</NAME>
          </OPERATION>
          <OPERATION operationID="getHost"
             comment="Get the host name where the process engine is running"
             transactional="no" returnType="java.lang.String">
             <NAME>getHost</NAME>
          </OPERATION>
        </OPERATIONS>
      </UnP-OPERATION-INTERFACE>
     </UnP-SERVICE-INTERFACE>
     <CONTEXT contextID="win">
      <PLATFORM processorArch="Pentium" os="Windows" version="NT 4">
      </PLATFORM>
      <BINARY path="S:\ApelV5\dev\classes\apel\RMI\ps\">
            RMI_ProcessEngine.class
      </BINARY>
     </CONTEXT>
    </MODEL>

    <MODEL modelID="CVS" client-server="yes" portable="no"
     persistent-state="yes" modifyFS="yes" generates-events="no"
     user-interaction="yes">
     <MODEL-DESC>Another well known version manager</MODEL-DESC>
     <WnP-SERVICE-INTERFACE>
      <COMMAND-INTERFACE>
         <COMMANDS>
            <COMMAND commandID="import"
                comment="create a new file" transactional="no">
```

```
                <NAME>cvs import</NAME>
                <ARGUMENT required="yes" type="path">chemin_module</ARGUMENT>
                <ARGUMENT required="yes" type="name">nom_branche</ARGUMENT>
                <ARGUMENT required="yes" type="name">nom_version</ARGUMENT>
                <ARGUMENT required="no" type="option">-d</ARGUMENT>
                <!-- Must be completed -->
            </COMMAND>
            <COMMAND commandID="export"
                comment="extract a copy of a module" transactional="no">
                <NAME>cvs export</NAME>
                <ARGUMENT required="no" type="option">-D</ARGUMENT>
                <!-- Must be completed -->
            </COMMAND>
            <COMMAND commandID="checkout"
                comment="checkout a file copy" transactional="no">
                <NAME>cvs checkout</NAME>
              <ARGUMENT required="no" type="option">-d</ARGUMENT>
                <!-- Must be completed -->
            </COMMAND>
            <COMMAND commandID="commit"
                comment="performs a new version" transactional="no">
                <NAME>cvs commit</NAME>
                <ARGUMENT required="yes" type="file">f1</ARGUMENT>
                <ARGUMENT required="no" type="option">-f</ARGUMENT>
                <!-- Must be completed -->
            </COMMAND>
            <COMMAND commandID="update"
                comment="update files" transactional="no">
                <NAME>cvs update</NAME>
                <ARGUMENT required="no" type="option">-A</ARGUMENT>
                <!-- Must be completed -->
            </COMMAND>
            <COMMAND commandID="history"
                comment="allow   to   visualize   all   base   actions   history"
transactional="no">
                <NAME>cvs history</NAME>
                <ARGUMENT required="no" type="option">-a</ARGUMENT>
                <!-- Must be completed -->
            </COMMAND>
         </COMMANDS>
         <InitC-COMMANDS>
            <COMMAND commandID="init"
                comment="New CVS base initialization" transactional="no">
                <NAME>cvs init</NAME>
          </COMMAND>
         </InitC-COMMANDS>
      </COMMAND-INTERFACE>
     </WnP-SERVICE-INTERFACE>
     <CONTEXT contextID="win">
      <PLATFORM     processorArch="Pentium"     os="Linux"     version="RedHat
6.2"></PLATFORM>
      <BINARY path="/usr/bin">cvs</BINARY>
     </CONTEXT>
    </MODEL>
    <!-- Other MODELs may be declared -->
  </MODELS>

  <INSTANCES>
    <INSTANCE instanceID="myRCS_70 modelID="RCSforWindows"
        locationID="verjus">
        <COMP-PATH>c:\\Program Files\\CSRCS\\</COMP-PATH>
    </INSTANCE>
    <INSTANCE instanceID="myCVS_39 modelID="CVS"
        locationID="zidane">
        <COMP-PATH>/usr/bin/</COMP-PATH>
    </INSTANCE>
    <INSTANCE instanceID="myPE_39 modelID="ProcessEngine"
        locationID="zidane">
        <COMP-PATH>s:\\ApelV5\\dev\\classes\\</COMP-PATH>
    </INSTANCE>
    <!-- To be completed -->
```

```
  </INSTANCES>

  <INSTANCES-CONTROL>
    <INSTANCE-CONTROL instanceID="myRCS_70" beginDate="02/03/2000"
        endDate="31/12/2001" deconnection="PROHIBITED" mobility="PROHIBITED"
        proxy="yes">
    </INSTANCE-CONTROL>
    <INSTANCE-CONTROL instanceID="myCVS_39" beginDate="02/03/2000"
        endDate="31/12/2001" deconnection="PROHIBITED" mobility="PROHIBITED"
        proxy="yes">
    </INSTANCE-CONTROL>
    <INSTANCE-CONTROL instanceID="myPE_39" beginDate="02/03/2000"
        endDate="31/12/2001" deconnection="PROHIBITED" mobility="PROHIBITED"
        proxy="no">
    </INSTANCE-CONTROL>
  </INSTANCES-CONTROL>

  <ROLE-COOPERATION>
    <ROLE-COOPERATION-DESC>
        How tools are involved in all roles
    </ROLE-COOPERATION-DESC>
    <ROLE-BEHAVIOR roleID="WorkspaceManager">
        <ROLE-PARTICIPANT>myRCS_70</ROLE-PARTICIPANT>
        <ROLE-PARTICIPANT>myCVS_39</ROLE-PARTICIPANT>
        <!-- May be completed -->
        <ROLE-BODY>
            <SELECT roleServiceID="productTransfert">
                <CONDITIONS>
                    <CONDITION not="no">
                        <OPERAND value="p" variable="yes"></OPERAND>
                        <OPERATOR type="CONTAINS"></OPERATOR>
                        <OPERAND value="spec" variable="no"></OPERAND>
                    </CONDITION>
                </CONDITIONS>
                <SERVICE-PERFORMERS>
                    <SERVICE-PERFORMER           instanceID=myCVS_39></SERVICE-
PERFORMER>
                </SERVICE-PERFORMERS>
            </SELECT>
            <SELECT roleServiceID="productTransfert" default="yes">
                <SERVICE-PERFORMERS>
                    <SERVICE-PERFORMER           instanceID=myRCS_70></SERVICE-
PERFORMER>
                </SERVICE-PERFORMERS>
            </SELECT>
        </ROLE-BODY>
     </ROLE-BEHAVIOR>
  </ROLE-COOPERATION>

  <PROXIES>
   <PROXY proxyID="RCSProxy_70" instanceID="myRCS_70" selfAdaptative="no">
    <MAPPINGS>
        <MAPPING roleServiceID="productTransfert">
            <P-PARTICIPANT destination="wrapper">checkIn</P-PARTICIPANT>
            <!--  The  'wrapper'  destination  refers  to  the  instance  tool's
wrapper -->
            <!-- and is used in the following piece of code -->
            <CODE-BODY>
                <CODE-FILE>
                    <!-- The file containing the PROPXY expressed in java -->
                    \\federation\\mappings\\RCSProxy.java
            </CODE-PIECE>
           <CODE-BODY>
        </MAPPING>
    </MAPPINGS>
   </PROXY>
   <PROXY proxyID="CVSProxy_39" instanceID="myCVS_39" selfAdaptative="no">
    <MAPPINGS>
        <MAPPING roleServiceID="productTransfert">
            <P-PARTICIPANT destination="wrapper">checkIn</P-PARTICIPANT>
```

```
            <!-- The 'wrapper' destination refers to the instance tool's
wrapper -->
            <!-- and is used in the following piece of code -->
            <CODE-BODY>
                <CODE-FILE>
                    <!-- The file containing the PROPXY expressed in java -->
                    \\federation\\mappings\\CVSProxy.java
            </CODE-PIECE>
           <CODE-BODY>
        </MAPPING>
    </MAPPINGS>
   </PROXY>
  </PROXIES>

</FEDERATION>
```

3 Traces obtenues lors de l'exécution complète du scénario de transfert de produit

Exécution avec contrôle lors des invocations et exécution d'aspects

```
C:\java\jdk\jdk1.3\bin\java.exe   federation.test.Bootstrap
Working Directory - X:\adele\data\Proto\v02\src\federation\
Class                                  Path                                  -
X:\adele\data\Proto\v02\classes;.;c:\java\ide\Kawa3.5\kawaclasses.zip;

 c:\java\jdk\jdk1.3\lib\tools.jar;c:\java\jdk\jdk1.3\jre\lib\rt.jar;c:\java\jd
k\jdk1.3\jre\lib\i18n.jar


  === The Application Common Universe (ACU) API is now created ...
 === The Right Control Server is started
 === Federation model is being loaded ...
 === The scenario is now reified : all objects have been created in the ACU
 === The Foundation Control is now started ...
 === The APEL Process Engine wrapper is now started
 === Federal Service register invocation has been caught by the Foundation
Control...
 === The current invocation is AUTHORIZED !!!
 === Tool ProcessEngine is now registered to the federation
 === The tool ProcessEngine has the ID tool-001
 === PE wrapper invokes the ACU ...
 === Federal Service getActivity invocation has been caught by the Foundation
Control...
 === The current invocation is AUTHORIZED !!!
 === The ACU Federal Service getActivity invocation has been caught by the
Control Foundation...
 === The Right Control Server is checking if the invocation is authorized or
not ...
 === A new invocation of the FEDERAL SERVICE : getActivity(java.lang.String)
 === The aspect trigger currently tested IS A method
 === The aspect trigger currently tested IS A method
 === The number of aspects THREAD for the trigger invocation getActivity@1 is
: 0
 === At least, one aspect thread is terminated
 === Before the Federal Service thread invocation
 === The FEDERAL SERVICE has been successfully executed
 === Federal Service terminateActivity invocation has been caught by the
Foundation Control...
 === The current invocation is AUTHORIZED !!!
 === The ACU Federal Service terminateActivity invocation has been caught by
the Control Foundation...
 === The Right Control Server is checking if the invocation is authorized or
not ...
```

```
 ===     A     new     invocation     of     the     FEDERAL     SERVICE     :
terminateActivity(java.lang.String)
 === The aspect trigger currently tested IS A method
 === The Federal Service currently invoked and the aspect's trigger currently
tested are different
 === The aspect trigger currently tested IS A method
 === The Federal Service currently invoked and the aspect's trigger currently
tested are identical
 => terminateActivity_A2  1
 === The aspect THREAD terminateActivity_A2 triggered by terminateActivity@2
is NOW invoked
 === The aspect terminateActivity_A2 is TERMINATED
 >>> In aspect body terminateActivity_A2 <<<
 >>> The ASPECT is invoking the role ...
 === The role WorkspaceManager is now invoked
 === ...and is identifiying which tool has to be invoked...
 === The role WorkspaceManager is invoking the RCS proxy
 === The RCS proxy is performing all necessary translations product->file
 === The RCS proxy is invoking the RCS wrapper ...
 === The RCS wrapper is calling the RCS tool for service request ...
 before exec de c:\Program Files\ComponentSoftware\CS-RCS\System\csrcs CheckIn
c:\temp\doc_de_specs.doc
 ok
 >>> The aspect is sending to the Control Foundation the normal termination
signal
 === Aspect terminateActivity_A2 has been removed from the aspects list !
 ===   The   number   of   aspects   THREAD   for   the   trigger   invocation
terminateActivity@2 is : 1
 === At least, one aspect thread is terminated
 === Before the Federal Service thread invocation
 ### Federal Service terminate successfully invoked !
 === The FEDERAL SERVICE has been successfully executed

 Process Exit...
```

Les traces de l'exécution montrent à la fois les contrôles effectués par la fondation de contrôle et à la fois le déclenchement d'un aspect associé au service fédéral "terminateActivity".

Exécution avec invocation interdite

```
C:\java\jdk\jdk1.3\bin\java.exe   federation.test.Bootstrap
Working Directory - X:\adele\data\Proto\v02\src\federation\

Class                                  Path                                   -
X:\adele\data\Proto\v02\classes;.;c:\java\ide\Kawa3.5\kawaclasses.zip;
c:\java\jdk\jdk1.3\lib\tools.jar;c:\java\jdk\jdk1.3\jre\lib\rt.jar;
c:\java\jdk\jdk1.3\jre\lib\i18n.jar


=== The Application Common Universe (ACU) API is now created ...
 === The Right Control Server is started
 === Federation model is being loaded ...
 === The scenario is now reified : all objects have been created in the ACU
 === The Foundation Control is now started ...
 === The APEL Process Engine wrapper is now started
 === Federal Service register invocation has been caught by the Foundation
Control...
 === The current invocation is AUTHORIZED !!!
 === Tool ProcessEngine is now registered to the federation
 === The tool ProcessEngine has the ID tool-001
 === PE wrapper invokes the ACU ...
 === Federal Service getActivity invocation has been caught by the Foundation
Control...
 === The current invocation is PROHIBITED !!!
 === Federal Service terminateActivity invocation has been caught by the
Foundation Control...
 === The current invocation is AUTHORIZED !!!
```

```
 === The ACU Federal Service terminateActivity invocation has been caught by
the Control Foundation...
 === The Right Control Server is checking if the invocation is authorized or
not ...

 java.lang.reflect.UndeclaredThrowableException:
 java.lang.reflect.InvocationTargetException:
 java.lang.NullPointerException
 at
federation.acu.fed.ACU_API_InvocationHandler.invoke(ACU_API_InvocationHandler.
java:92)
 at $Proxy1.terminateActivity(Unknown Source)
 at java.lang.reflect.Method.invoke(Native Method)
 at
federation.wrapper.Wrapper_InvocationHandler.invoke(Wrapper_InvocationHandler.
java:59)
 at $Proxy0.terminateActivity(Unknown Source)
 at federation.wrapper.PEWrapper.start(PEWrapper.java:54)
 at federation.test.Bootstrap.main(Bootstrap.java:62)
 Exception in thread "main"


 Process Exit...
```

Le moteur de processus n'a pas l'autorisation d'invoquer le service fédéral "getActivity(String)". Ce service fédéral devrait normalement retourner une référence sur l'activité de spécification. Cette référence est utilisée dans l'appel du service fédéral suivant (qui lui est autorisé). Comme la référence n'a pas été obtenue, le programme s'interrompt à la suite d'une erreur.

Exécution sans déclenchement d'aspect

```
C:\java\jdk\jdk1.3\bin\java.exe   federation.test.Bootstrap
Working Directory - X:\adele\data\Proto\v02\src\federation\

Class                                   Path                                    -
X:\adele\data\Proto\v02\classes;.;c:\java\ide\Kawa3.5\kawaclasses.zip;
c:\java\jdk\jdk1.3\lib\tools.jar;c:\java\jdk\jdk1.3\jre\lib\rt.jar;
c:\java\jdk\jdk1.3\jre\lib\i18n.jar

=== The Application Common Universe (ACU) API is now created ...
 === The Right Control Server is started
 === Federation model is being loaded ...
 === The scenario is now reified : all objects have been created in the ACU
 === The Foundation Control is now started ...
 === The APEL Process Engine wrapper is now started
 === Federal Service register invocation has been caught by the Foundation
Control...
 === The current invocation is AUTHORIZED !!!
 === Tool ProcessEngine is now registered to the federation
 === The tool ProcessEngine has the ID tool-001
 === PE wrapper invokes the ACU ...
 === Federal Service getActivity invocation has been caught by the Foundation
Control...
 === The current invocation is AUTHORIZED !!!
 === The ACU Federal Service getActivity invocation has been caught by the
Control Foundation...
 === The Right Control Server is checking if the invocation is authorized or
not ...
 === A new invocation of the FEDERAL SERVICE : getActivity(java.lang.String)
 === The number of aspects THREAD for the trigger invocation getActivity@1 is
: 0
 === At least, one aspect thread is terminated
 === Before the Federal Service thread invocation
 === The FEDERAL SERVICE has been successfully executed
```

```
 === Federal Service terminateActivity invocation has been caught by the
Foundation Control...
 === The current invocation is AUTHORIZED !!!
 === The ACU Federal Service terminateActivity invocation has been caught by
the Control Foundation...
 === The Right Control Server is checking if the invocation is authorized or
not ...
 ===     A     new     invocation     of     the     FEDERAL     SERVICE     :
terminateActivity(process.apel.Activity, java.lang.String)
 ===   The   number   of   aspects   THREAD   for   the   trigger   invocation
terminateActivity@2 is : 0
 === At least, one aspect thread is terminated
 === Before the Federal Service thread invocation
 ### Federal Service terminate successfully invoked !
 === The FEDERAL SERVICE has been successfully executed

 Process Exit...
```

Les traces de l'exécution montrent qu'aucun aspect n'a été déclenché. Le déroulement s'est effectuée comme si la fondation de contrôle n'existait pas dans le sens où il n'y avait ni interdiction, ni aspect...Le comportement de cette fédération suit le paradigme de "l'anarchie".

Annexe C : Mécanismes de contrôle des fédérations d'outils

Un des objectifs fixé dans le cadre de cette thèse est de cacher l'ensemble de la logique de fonctionnement de la fédération (c'est-à-dire cacher la fondation de contrôle) aux outils participants. Du côté des outils (et de leurs façades respectives), seule l'API de l'univers commun de l'application, constituée des services fédéraux est "visible". Le reste, c'est-à-dire les mécanismes de contrôle doivent être entièrement transparents.

Sur invocation d'un service fédéral de l'UCA effectuée par un outil (1) de la fédération (voir Figure 1) et en fonction du contrôle défini par le concepteur de la fédération, plusieurs cas peuvent se présenter. Ces cas, que nous allons détailler, sont exclusifs les uns des autres pour une même invocation (c'est-à-dire un couple formé par un participant à la fédération et un service fédéral donné).

1 Invocation interdite

Cette situation se produit (Figure 1) lorsque l'outil n'a pas le droit d'invoquer le service fédéral en question. Cela signifie que les rôles que l'outils endosse contiennent une interdiction portant sur l'invocation du service fédéral indiqué.

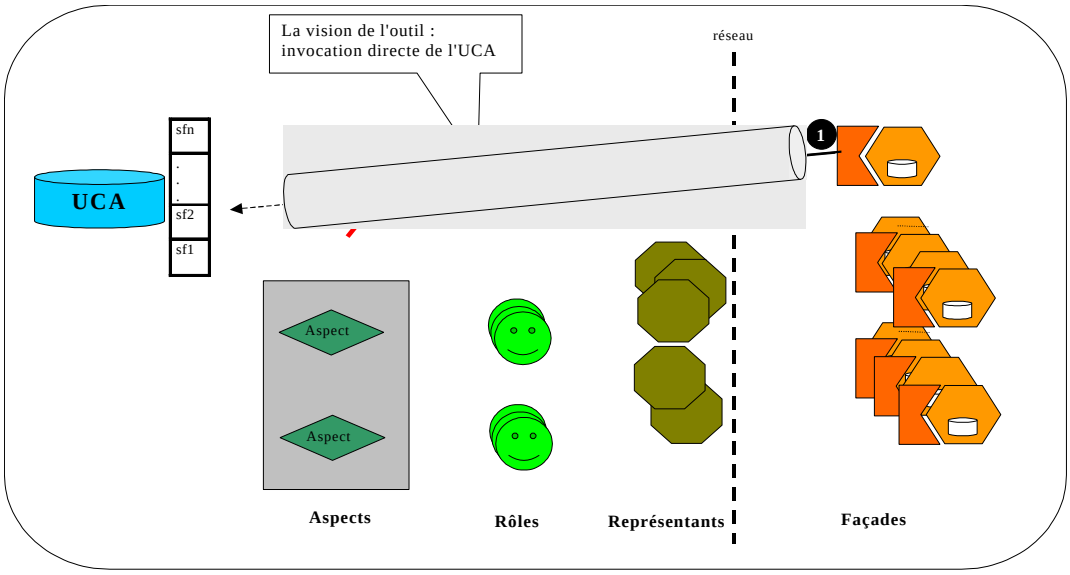

Figure 1 Interdiction d'utiliser un service fédéral

La façade de l'outil invoque un service fédéral (1). Cette invocation est contrôlée par la fondation de contrôle qui relève une interdiction (définie dans le modèle de fédération). L'invocation est alors stoppée (2) puisque étant interdite.

Nous notons que cette situation nécessite la présence de la fondation de contrôle.

2 Invocation autorisée (ou non contrôlée) sans exécution d'aspect

Cette situation caractérise l'exécution normale d'un service fédéral de l'UCA sans déclenchement d'aspect : aucun aspect défini par le concepteur n'est associé au service fédéral en question.

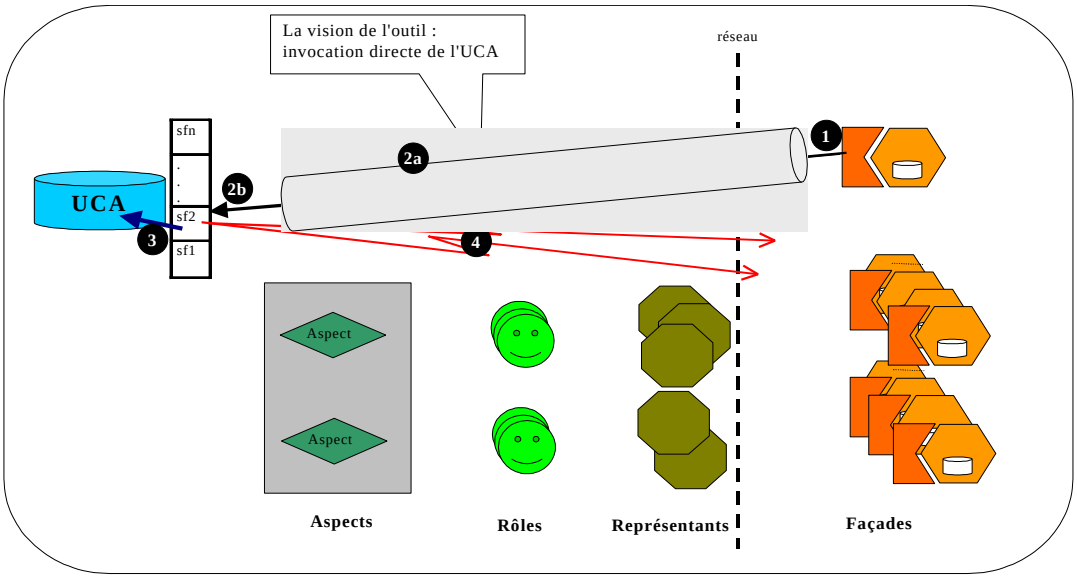

Figure 2 Invocation d'un service fédéral

Nous notons que cette situation ne nécessite pas forcément la présence de la fondation de contrôle.

En effet, si la fondation de contrôle est présente, le contrôle de l'autorisation de l'invocation du service fédéral par l'outil est effectué par la fondation de contrôle (2a). Cette dernière n'ayant pas constaté d'interdiction, autorise l'invocation (2b), qui est réalisée normalement (3). Puis, les notifications sont envoyées aux souscripteurs (4) suivant les autorisations mises en place. Dans le cas où la fondation de contrôle est absente, aucune procédure de vérification des droits (2a) n'est réalisée et l'invocation est normalement réalisée (3)...

3 Invocation autorisée avec association d'aspect(s)

Cette situation (Figure 3) caractérise le fait que l'outil a effectivement le "droit" d'invoquer le service fédéral en question :

1) l'invocation initiale est détournée (2b), en parallèle vers chacun des aspects associés au service fédéral invoqué (2a) ;
2) chacun de ces aspects s'exécute et invoque les rôles impliqués dans l'aspect (3) ;
3) les rôles, à leur tour, vont invoquer les outils "pertinents" (4) par rapport à la situation (soit directement, soit au travers de leur représentant) ;
4) les représentants sollicités vont effectuer les conversions nécessaires et vont invoquer les outils au travers de leurs façades (5) ;
5) dès qu'un des aspects a terminé son exécution, le service fédéral est effectivement invoqué (6a et 6b) sur l'UCA pour que ce dernier soit dans un état cohérent (reflète les changements) avec les actions "réelles" ;
6) les notifications sont ensuite envoyées aux souscripteurs (7) suivant les autorisations mises en place.

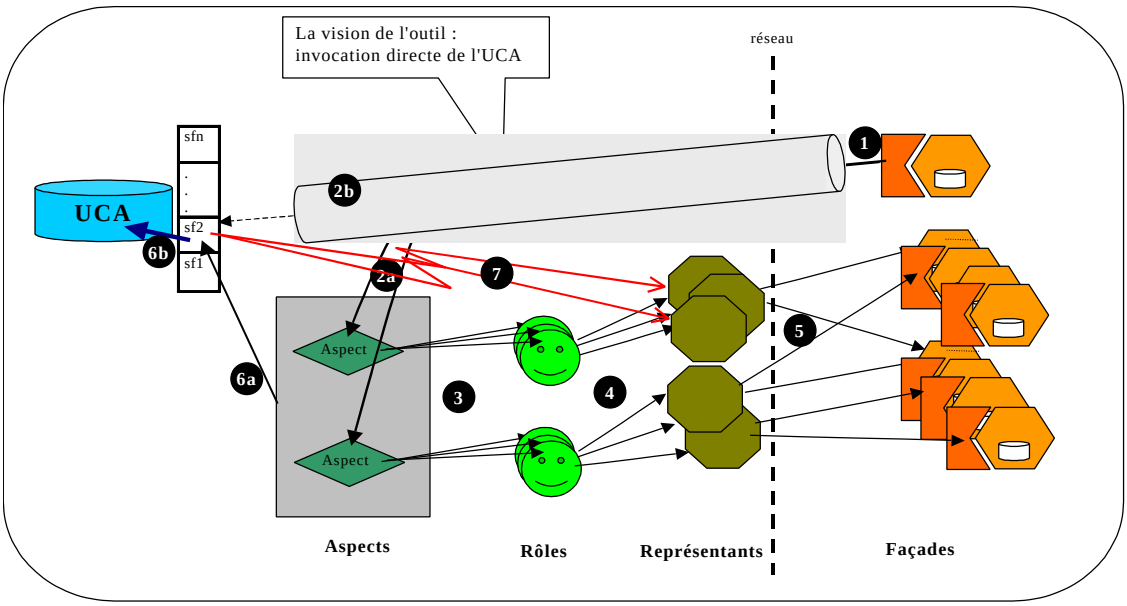

Figure 3 Principe de fonctionnement complet d'un service fédéral avec des aspects associés

En ce qui concerne le scénario de transfert de produit, la façade du moteur de processus "veut" invoquer le service fédéral dont l'identifiant est "terminateActivity". Cet appel (invocation) va être capturé par la fondation de contrôle qui va le dérouter au niveau de chaque aspect avec lequel une association a été définie. Dans notre cas, nous avons défini un seul aspect (cas simple). Cet aspect comprend et interprète la définition de la coordination des différents rôles impliqués (rôle unique : *WorkspaceManager*) ; ce dernier va solliciter les outils. Dès qu'un des aspects, parmi l'ensemble des aspects associés au service fédéral en question, est terminé, le service fédéral est effectivement invoqué (6a).

4 Mise en place des paradigmes

Soit *I* l'ensemble des instances (les participants) de la fédération. Soit *S* l'ensemble des services fédéraux de l'UCA. Nous avons le produit cartésien *I x S* qui définit l'ensemble des invocations possibles concernant les services fédéraux et de la part des instances de la fédération. Ce produit cartésien est un ensemble de couples *(i, s)* tels que *i* est un élément de *I* et s un élément de *S*.

Cela veut dire que pour tout couple *(i, s)* de *I x S*, il est possible de définir un mode de fonctionnement particuliers tel que ceux décrits dans les sections 1., 2., 3. de cette annexe. D'autre part cela permet, pour tout service fédéral *s* de *S* de lui associer une politique de contrôle plus ou moins forte. Cette "dureté" dans le contrôle est donnée en associant, au besoin (ou non), un ensemble d'aspects et en utilisant certaines propriétés pour les aspects lorsque cela est possible (mode transactionnel).

L'ensemble des modes ainsi définis pour chaque aspect associés aux services fédéraux caractérise le mode de fonctionnement global de la fédération.

Il est alors possible d'implémenter une fédération faiblement contrôlée ou, au contraire fortement contrôlée, balayant le spectre dont les extrémités sont "l'anarchie" et la "dictature" [Estublier et al. 1998b, Estublier et Verjus 1999, Estublier et al. 2001a].

Par rapport aux travaux existants dans ce domaine, nous permettons de passer du niveau de l'architecture générale qui implémente un mode de fonctionnement et un seul [Estublier et al. 1998a] à une architecture pouvant supporter différents modes de fonctionnement, en intervenant au niveau de granularité constitué des couples *(i, s)* et des aspects.

5 Configurations de la fondation de contrôle pour supporter différentes "politiques"

Dans [Estublier et al. 2001a, Estublier et al. 2001b], différentes "politiques" peuvent être mises en place. Nous reprenons le cas des politiques selon le paradigme de l'"anarchie" ou "anarchie contrôlée", et celle de la "dictature" qui vont être appliquées au cas du scénario de transfert de produit.

L'anarchie

Pour cela, le concepteur de la fédération va définir que le rôle *WorkspaceManager* va devoir s'abonner au service fédéral "terminateActivity". A partir de cette définition, la fondation de contrôle va automatiquement souscrire l'ensemble des outils jouant le rôle *WorkspaceManager* au service fédéral "terminateActivity". Les représentants de ces derniers vont recevoir les notifications se rapportant à l'utilisation de ce service fédéral, vont effectuer les conversions nécessaires puis vont à leur tour notifier les façades outils. A ce stade du processus de contrôle, les représentants n'ont plus à s'occuper[72] du devenir de leurs notifications (cas "anarchique"). Les façades des outils sont alors "libres" de réagir aux notifications ou non. Mais l'absence de réactions n'a aucun impact sur le comportement de la fédération. En d'autres termes, si les outils ne font rien à leur niveau, la fondation de contrôle n'en saura rien et continuera son exécution normalement.

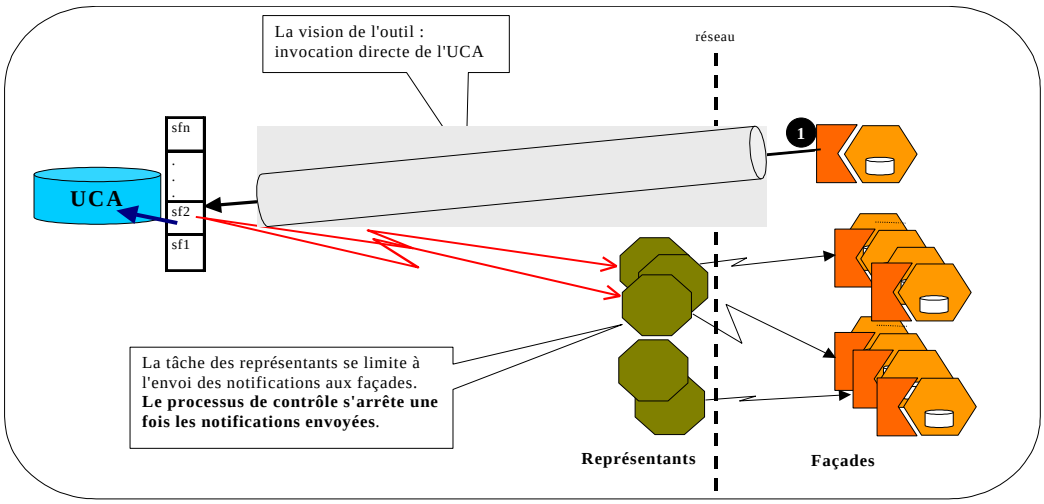

Figure 4 Processus de contrôle dans le cas de "l'anarchie"

Dans le cas où l'outil n'aurait pas de représentant associé, ce serait la façade de l'outil qui recevrait directement les notifications.

Dans cette configuration, aucun aspect n'a été défini. La description du rôle *WorkspaceManager* ne comporte que le seul devoir de s'abonner aux notifications du service fédéral "terminateActivity". Généralement, des rôles dont la définition en comporte que des souscriptions caractérisent un fonctionnement davantage "anarchique". Une telle fédération est simple à définir.

Dans le quatrième chapitre, nous avions montré qu'une fédération complètement "anarchique" n'impose pas la présence d'une fondation de contrôle. C'est en effet exact et la fédération décrite ci-dessus n'exige pas d'avoir une fondation de contrôle. Cependant, la présence de la fondation de contrôle peut s'avérer utile, non seulement pour contrôler la fédération, mais plus simplement (voire uniquement) parce qu'elle comprend l'UCC contenant le modèle de

[72] Voir encadré dans la figure 4.

fédération... Et à partir duquel certaines vérifications pourront être effectuées lorsque la question de l'évolution de fédération d'outils sera abordée.

D'autre part, le simple fait de contrôler les invocations des services fédéraux nécessite la présence de la fondation de contrôle. Dans ce cas, une telle fédération suit un mode qualifié "d'anarchique contrôlé" [Estublier et al. 2001b].

La dictature

Dans ce cas, le concepteur de la fédération va définir un aspect (au moins un). Cet aspect va être associé au service fédéral "terminateActivity" ; cette association définit que sur invocation du service fédéral, l'aspect est prioritairement exécuté. L'aspect en question appelle/invoque, lors de son exécution, le rôle *WorkspaceManager* au travers du service `transfert(Product p)`. Ce dernier transmet l'invocation au représentant de l'outil concerné (RCS ou CVS suivant que le document est un document de spécification ou non). Le représentant va solliciter l'outil au travers de sa façade en attendant un retour[73] de cette dernière (contrairement au cas précédent).

Dans cette configuration, le processus d'exécution du service fédéral est bien contrôlé (par la fondation de contrôle) de l'invocation du service fédéral par l'outil, à l'invocation du service fédéral sur l'UCA. Ce cas est un peu plus complexe que le précédent à définir. Un aspect doit être défini. Sa définition comprend :

- une association avec le service fédéral "terminateActivity" ;
- l'invocation du service abstrait `transfert(Product p)` du rôle *WorkspaceManager*.

Par contre, le rôle *WorkspaceManager* ne comprend plus la même définition. Dans ce cas, il doit fournir un service (`transfert(Product p)`)permettant de transférer le produit en remplacement de la souscription du cas précédent qui, elle, n'existe plus.

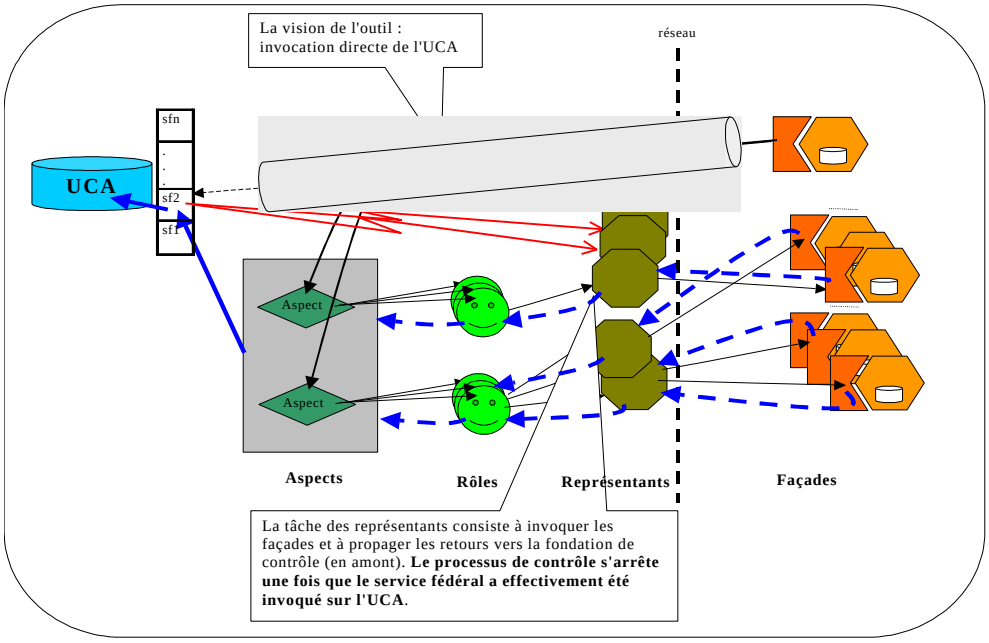

Figure 5 Processus de contrôle dans le cas de la "dictature"

Dans le cas où l'outil n'aurait pas de représentant associé, ce serait la façade de l'outil qui recevrait directement les invocations émanant de la fondation de contrôle et qui devrait rendre "la main" à la fondation de contrôle.

[73] Voir encadré dans la figure 5.

Annexe D : Publications réalisées dans le cadre de la thèse

1 Conférences avec actes

[1] H. Verjus, "Vers une Fédération d'Environnements Centrés Processus", Journées Jeunes Doctorants, MATIS'98, Archamps, France, mars 1998.

[2] H. Verjus, F. Oquendo, “Fédérations d’environnements centrés processus logiciels : une approche basée composants”, Proceedings of the 11th International Conference on Software Engineering and its Applications (GL’98), Paris, décembre 1998.

[3] I. Alloui, S. Cimpan, F. Oquendo, H. Verjus « ALLIANCE : an Agent-based Case Environment for Enterprise Process Modelling, Enactment and Quantitative Control », Proceedings of the 1st International Conference on Enterprise Information Systems, 27-30/03/1999, Setubal, Portugal.

[4] I. Alloui, S. Cimpan, F. Oquendo, H. Verjus « Tuning a Fuzzy Control System for Software Intensive Processes via Simulations », Proceedings of the IASTED International Conference on Modeling and Simulation, Philadelphie PA, USA, mai 1999

[5] I. Alloui, S. Cimpan, F. Oquendo, H. Verjus, « A Fuzzy Sets based Mechanism Allowing the Tuning of a Software Intensive Processes Control System via Multiple Simulations », Proceedings of the AMSE International Conference on Modelling and Simulation MS’99, Santiago de Compostela, Espagne, mai 1999.

[6] I. Alloui, S. Cimpan, F. Oquendo, H. Verjus, "Software Agents for bringing cooperating software-intensive processes under quantitative control using fuzzy sets", Proceedings of Many Facets of Process Engineering MFPE’99, Tunisia, mai 1999.

[7] I. Alloui, S. Cimpan, F. Oquendo, H. Verjus, "ALLIANCE: A Software Framework for Software-intensive Process Modeling, Enactment and Fuzzy Control", Proceedings of the Integrated Design and Process Science IDPT’99, tenue conjointement à IDPT'2000, Dallas, Texas, EU, 2000.

[8] T. Bolusset, F. Oquendo, H. Verjus, "Software Component-based Federations are Software Architectures too", Proceedings of the International Process Technology Workshop, IPTW'99, Villars-de-Lans, France, septembre 1999.

[9] G. Cugola, P. Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, "Support for Software Federations: the PIE Platform", Proceedings of the 7th European Workshop on Software Process Technology, EWSPT’2000 (conference track), Autriche, février 2000.

[10] G. Cugola, P.Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Riviere, and H. Verjus, "Customizing the behavior of middleware: the PIE approach" in Proceedings of the Workshop on Reflective Middleware (RM2000), New York (USA), 7-8 avril, 2000.

[11] J. Estublier, H. Verjus, P.Y. Cunin, "Building Software Federation", Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Las-Vegas, US, 25-28 juin, 2001.

[12] J. Estublier, H. Verjus, P.Y. Cunin, "Designing and Building Software Federations", Proceedings of 1st Conference on Component Based Software Engineering, (CBSE), Warsaw, Poland, 4-6 septembre 2001.

[13] J. Estublier, H. Verjus, P.Y. Cunin., "Modelling and Managing Software Federations", European Conference on Software Engineering (ESEC), Wien, Austria. 10-14 septembre, 2001.

2 Chapitre d'ouvrage

[14] I. Alloui, S. Cimpan, F. Oquendo et H. Verjus : "Alliance: An Agent-Based CASE Environment for Enterprise Process Modelling, Enactment and Quantitative Control", Enterprise Information Systems, Joaquim Filipe (Ed.), KLUWER ACADEMIC PUBLISHERS. (à paraître).

3 Rapports internes LLP/CESALP/US

[15] H. Verjus, "Interopérabilité: état de l'art, mécanismes, travaux et orientations", Rapport Interne N°0000601, 20/12/1998, LLP/CESALP/US.

[16] H. Verjus, "Notice de configuration et d'installation des composants logiciels: monitoring support, decision support, change support", Rapport Interne N°0000602, 17/03/1999, LLP/CESALP/US.

4 Livrables de projets

PIE (Process Instance Evolution) – Projet Européen ESPRIT N° 24840 (PHASE I)

[17] S. Cîmpan, H. Verjus "Requirements and approach for monitoring support and interfaces definitions", Technical Report, MS-DDOC, PIE ESPRIT LTR IV Project No. 24840, mai 1998.

[18] S. Cîmpan, H. Verjus "Requirements and approach for decision support and interfaces definitions", Technical Report, MS-DDOC, PIE ESPRIT LTR IV Project No. 24840, mai 1998.

[19] H. Verjus et S. Cîmpan "Technical Description of the Monitoring and Decision Support PIE Component Prototypes", Technical Report, MS-DPRO & DS-DPRO, PIE ESPRIT LTR IV Project No. 24840, avril 1998.

PIE (Process Instance Evolution) – Projet Européen ESPRIT N° 34840 (PHASE II)

[20] J. Estublier and H. Verjus, “Definition of the behaviour paradigms of a heterogeneous federation of evolving process components”, PIE LTR ESPRIT Project 34840, Deliverable D2.01, juin 1999.

[21] Gianpaolo Cugola, Pierre-Yves Cunin, Jacky Estublier, Alfonso Fuggetta, François Pacull, Michel Rivière, Hervé Verjus, "Concepts and paradigms for the design of evolving, distributed and mobile processes", PIE LTR ESPRIT Project 34840, Deliverable D2.03, novembre 1999.

[22] Gianpaolo Cugola, Pierre-Yves Cunin, Jacky Estublier, Alfonso Fuggetta, François Pacull, Michel Rivière, Hervé Verjus, "Concepts and paradigms for the design of evolving, distributed and mobile processes ", PIE LTR ESPRIT Project 34840, Deliverable D2.04, mars 2000.

[23] M. Becchi, G. Cugola, P.Y. Cunin, J. Estublier, A. Fuggetta, F. Pacull, M. Rivière, H. Verjus, "Architecture approach and formalization", PIE LTR ESPRIT Project 34840, Deliverable D2.05, octobre 2000.

[24] J. Estublier and H. Verjus, “Definition and design of PSS elements to support evolution of heterogeneous, distributed and mobile processes in the PIE system", PIE LTR ESPRIT Project 34840, Deliverable D2.06, janvier 2001.

EDF/DER (Contrat d'études, recherches et développement)

[25] H. Verjus, F. Oquendo, C. Chevenier, "Etat de l’art sur l’échange de données informatisées : concepts, mécanismes, travaux et tendances sur l'interopérabilité", Contrat d'études, recherche et développement, Electricité de France, Echanges de données informatisées pour des outils relevant de l'ingénierie système, livrable L3.2, mars 2000.