Table des matières

Table des figures

Chapitre I

Introduction générale

I.1. Contexte général

Le développement fulgurant des technologies de l'information, des réseaux et des moyens de communication est à l'origine d'une expansion formidable de l'Internet. Nous traversons une nouvelle ère où la démocratisation de l'Internet et le passage à la société de l'information conduisent à un processus de globalisation (ou mondialisation) incitant à la mise en relation des personnes et à la constitution d'équipes virtuelles réparties partout dans le monde. C'est ainsi l'ère du Travail Coopératif Assisté par Ordinateur (TCAO), caractérisée par le travail en groupes à distance, l'échange d'information et la mobilité.

Le TCAO est le domaine qui étudie la conception, la mise en oeuvre et l'utilisation d'applications collaboratives, également appelées collecticiels. Les collecticiels sont des systèmes coopératifs qui permettent à plusieurs utilisateurs de travailler ensemble par l'intermédiaire d'une infrastructure technologique (un intranet ou l'Internet). Différents types de collecticiels ont été développés : des outils de communication, d'échange ou de partage de données, *etc*. La vidéoconférence en est un exemple, tout comme les tableaux blancs partagés et les éditeurs coopératifs qui permettent de travailler à plusieurs simultanément sur un même document. Ces outils représentent tant des produits académiques que commerciaux.

Malheureusement, les collecticiels actuels font face à une diffusion limitée parmi leurs utilisateurs potentiels [Palen-03]. La recherche dans le domaine du TCAO s'interroge toujours sur les fondements à adopter pour le développement d'applications collaboratives, ceci dans le but de pouvoir proposer enfin des systèmes capables de supporter et de répondre au mieux aux besoins des utilisateurs. C'est dans ce contexte général que se situe notre problématique de recherche.

I.2. Problématique abordée

Le TCAO se trouve à l'intersection de différents domaines de recherche, dans le contexte de l'informatique ainsi que dans celui des sciences humaines. Cette multidisciplinarité fait apparaître des problèmes de différentes natures.

Sur le plan technique, il faut considérer qu'un collecticiel est une application interactive, multi-utilisateurs et répartie. Ainsi, différents aspects tels que l'accès concurrent aux ressources partagés, le contrôle d'accès par rapport aux droits des utilisateurs, la synchronisation de la communication réseau, entre autres, doivent être pris en compte par ce type d'applications.

Sur le plan sociologique, les collecticiels doivent prendre également en compte des aspects plus "humains" de la coopération. Il faut considérer, par exemple, comment l'activité collaborative peut être démarrée, c'est-à-dire, comment les utilisateurs peuvent se mettre à collaborer. La gestion des rôles des utilisateurs dans l'accomplissement d'une tâche collaborative, ainsi que celle de la conscience de groupe sont également importantes. Une autre question à résoudre est comment accorder des modes de travail individuel et collaboratif.

Outre la multidisciplinarité du domaine, il ne faut pas oublier qu'il s'agit de supporter des interactions dirigées par des utilisateurs. Les activités collaboratives impliquent différentes équipes et organisations exécutant différentes tâches, demandant chacune des fonctionnalités différentes. Un autre problème fondamental est celui posé par des besoins utilisateurs qui sont souvent imprévisibles – des besoins qui émergent au cours même de l'activité collaborative.

Ainsi, la conception des outils informatiques pour le TCAO s'avère une tâche intrinsèquement difficile. Des fortes exigences en terme d'évolutivité, de flexibilité et d'extensibilité pour les applications collaboratives sont imposées. Le but est de concevoir des collecticiels capables de supporter différents scénarios de collaboration tout en s'adaptant à des besoins particuliers des utilisateurs.

Les développeurs d'applications collaboratives se sont orientés vers différentes approches visant le développement de collecticiels. Nous pouvons distinguer trois approches principales : développement *from scratch* ; développement s'appuyant sur des boîtes à outils ; et développement s'appuyant sur des *frameworks* et des plates-formes.

Alors qu'un développement *from scratch* d'un système est acceptable pour des collecticiels spécifiques (c'est-à-dire, offrant des fonctionnalités collaboratives bien précises), il est admis que la conception logicielle d'une application collaborative plus complexe (intégrant différentes fonctionnalités, devant être en même temps extensibles) doit s'appuyer sur d'autres solutions, logicielles ou conceptuelles : des boîtes à outils, des *frameworks* et des plates-formes de développement.

Si d'un côté l'utilisation de boîtes à outils s'avère simple et rapide, le degré d'extensibilité obtenu est très restreint. Quant aux *frameworks* et aux plates-formes existants, l'extensibilité est effectivement prise en compte. Par contre, la possibilité d'intégrer de nouveaux composants est souvent contraignante puisque ces derniers doivent suivre un schéma d'interaction imposé à l'avance par le système en question. Il est donc difficile de pouvoir intégrer des composants ou d'autres applications collaboratives déjà disponibles sur le marché.

Indépendamment de l'approche de développement choisie, la conception de nouvelles applications collaboratives qui soient vraiment flexibles et extensibles (sans imposer de contraintes vis-à-vis des composants pouvant être intégrés) se présente comme un grand défi pour les développeurs. Ainsi, dans la pratique, les "environnements collaboratifs" actuels sont composés par les utilisateurs eux-mêmes – ils choisissent, selon leurs besoins, différentes applications offrant des fonctionnalités spécifiques, en les exécutant en même temps mais indépendamment les unes des autres. Néanmoins, l'intégration de ces applications collaboratives pourrait être très bénéfique aux utilisateurs. Différentes fonctionnalités pourraient être combinées et contrôlées de manière dynamique.

Notre objectif, sujet de notre travail de thèse, est ainsi de définir un environnement d'intégration pour des applications collaboratives existantes. De nombreux travaux se sont déjà focalisés sur cette problématique : l'intégration d'applications. Notamment dans le domaine des applications réparties, où le spectre va de solutions à base d'intergiciels (tels que CORBA et DCOM) à l'utilisation de nouvelles technologies de service Web.

Ces travaux représentent effectivement des réponses techniques au problème d'intégration, permettant aux applications de communiquer et d'échanger des informations. Ces travaux visent néanmoins à maîtriser uniquement ce qui est appelé l' "interopérabilité syntaxique" [Lewis-04], ou "intégration de surface" [Iqbal03]. Pour aboutir à une vraie intégration, il faut que les applications s'accordent par rapport à la sémantique de leurs échanges et la façon dont elles doivent réagir. Une telle intégration sémantique peut s'avérer être très complexe lorsqu'on envisage l'intégration d'applications développées par des tiers.

Ainsi, dans le cadre de cette thèse, nous proposons un environnement d'intégration qui, en s'appuyant sur une approche faiblement couplée, permet de réaliser l'intégration d'applications collaboratives existantes tout en évitant de considérer des détails internes à ces applications. Malgré son aspect faiblement couplé, cet environnement surpasse une simple intégration de surface, en proposant des mécanismes permettant d'associer à une session collaborative intégrée la sémantique désirée de cette intégration.

I.3. Principales contributions de la thèse

Plusieurs contributions peuvent être mises en avant au niveau de cette thèse, parmi lesquelles nous tenons à souligner :

1. la proposition d'un cadre général d'intégration s'appuyant sur une approche faiblement couplée, dont l'originalité se caractérise par son aspect hybride combinant des interfaces de services Web et un système de notification d'événements, permettant de réaliser l'intégration d'applications collaboratives développées par des tiers ;
2. la définition d'une méthode graphique pour la spécification de politiques de collaboration (correspondant à un ensemble de règles dont la sémantique est fondée sur des réseaux de Petri) permettant de définir la sémantique de l'intégration de différentes applications collaboratives utilisées dans le contexte d'une *SuperSession* ;
3. la définition d'une architecture qui, reposant sur une solution s'appuyant sur des *Wrappers*, prévoit l'intégration d'applications présentant différents profils de distribution (c'est-à-dire, client/serveur, multiserveurs et P2P), et qui, en même temps, définit une infrastructure complètement répartie pour l'exécution des sessions collaboratives intégrées (*i.e.* des *SuperSessions*) ;
4. une approche conceptuelle de type "top-down" pour modéliser l'architecture de notre système, LEICA, en nous appuyant sur le profil UML/SDL supporté par l'outil *TAU G2* de *Telelogic* ;
5. l'implémentation d'un prototype de LEICA, qui met en œuvre les principaux modules prévus dans l'architecture à ce jour, et qui supporte une grande partie des fonctionnalités décrites dans ce mémoire de thèse.

I.4. Plan de thèse

Ainsi notre thèse vise à proposer LEICA, un environnement faiblement couplé pour l'intégration d'applications collaboratives. En parcourant ce mémoire de thèse, le lecteur pourra découvrir les différents aspects de notre proposition. Le mémoire est structuré en cinq chapitres principaux, plus cette introduction et une conclusion générale.

Le premier chapitre, intitulé *Etat de l'art*, présente un état de l'art de notre problématique de recherche. Nous introduisons des concepts fondamentaux, termes et classifications afin de présenter le contexte de nos travaux. Dans ce contexte, nous abordons la problématique liée au développement d'applications collaboratives, notamment en ce qui concerne le besoin de disposer des environnements flexibles et intégrés. Nous analysons ainsi différentes approches de développement d'applications collaboratives ainsi que leurs limitations pour concevoir ces applications en tant qu'environnements intégrés. Nous achevons ce chapitre en présentant des solutions pour l'intégration d'applications, et en particulier, des solutions plus spécifiques dans le domaine du TCAO, tout en positionnant les idées qui ont été à l'origine de LEICA.

Le deuxième chapitre, intitulé *L'Approche d'intégration initiale*, décrit le point de départ qui a servi de base à la réalisation de nos travaux. Nous montrons comment lors de notre participation au Projet Européen Lab@Future, nous avons été confrontés à la problématique d'intégration d'applications. Nous présentons dans ce chapitre le CIE, un environnement que nous avons initialement développé dans le but de proposer une solution générale d'extension d'un CVE (un environnement virtuel collaboratif partageant des mondes en réalité virtuelle) à travers l'intégration d'autres outils de collaboration.

Le troisième chapitre, intitulé *L'environnement d'intégration : LEICA*, présente les principaux concepts sur lesquels s'appuie LEICA, l'environnement d'intégration proposé dans le cadre de cette thèse. Nous introduisons dans ce chapitre l'approche générale d'intégration faiblement couplée adoptée par l'environnement. Nous présentons également la définition d'une *SuperSession*, l'abstraction de base permettant à LEICA de gérer une session collaborative intégrée. Nous expliquons dans la suite le processus de configuration des *SuperSessions*, en détaillant la spécification des politiques de collaboration. A la fin de ce chapitre nous définissons, à l'aide des réseaux de Petri, la sémantique des règles définissant la politique de collaboration d'une *SuperSession*.

Le quatrième chapitre, intitulé *L'architecture de LEICA*, présente une description détaillée, bien qu'informelle, de l'architecture de LEICA permettant de mettre effectivement en œuvre l'approche générale d'intégration proposée dans le chapitre précédent. Nous présentons ainsi une vue globale de l'architecture en décrivant ses principaux modules ainsi que leurs rôles respectifs lors de l'exécution de LEICA. Partant de cette description intuitive de l'architecture, nous proposons une démarche pour la formaliser au moyen d'une modélisation basée sur le profil UML/SDL supporté par l'outil *TAU G2* de *Telelogic*. Nous détaillons l'architecture au moyen de différents diagrammes UML de ce profil et cherchons à valider l'architecture en comparant les traces de simulation obtenues avec les scénarios (diagrammes de séquences) élaborés au début de la phase de conception de notre système.

Le cinquième chapitre, intitulé *Implémentation et déploiement de LEICA*, présente les différentes technologies logicielles utilisées pour l'implémentation du prototype de LEICA. Il détaille ensuite l'implémentation réalisée, et les principaux modules implémentés sont décrits. Nous abordons ensuite l'intégration de deux applications collaboratives à l'environnement et décrivons la *SuperSession* créée dans le but de tester cette intégration.

Finalement, un dernier chapitre du manuscrit présente une conclusion générale de ce travail et définit quelques perspectives de continuation.

Chapitre II

État de l'art

II.1. Introduction

Dans ce chapitre nous présentons un état de l'art des applications destinées à offrir un support au travail coopératif.

Dans un premier temps, nous introduisons des concepts fondamentaux, termes et classifications afin de présenter le contexte de nos travaux.

Dans un deuxième temps, nous abordons la problématique liée au développement d'applications collaboratives. Nous présentons d'abord les difficultés auxquelles sont confrontés les développements pour concevoir des applications collaboratives qui répondent aux besoins divers des utilisateurs. Dans ce contexte la nécessité de disposer d'environnements intégrés apparaît clairement vue la difficulté d'anticiper de manière précise les besoins utilisateurs. Nous présentons ainsi différentes approches de développement d'applications collaboratives ainsi que leurs limitations pour concevoir ces applications en tant qu'environnements intégrés.

Finalement, la troisième partie de ce chapitre aborde la même approche que celle adoptée par LEICA. Cette approche consiste à intégrer des applications collaboratives existantes (et originellement indépendantes) pour aboutir à des environnements collaboratifs intégrés. Nous présentons des solutions générales pour réaliser l'intégration de ces applications, puis abordons des solutions plus spécifiques pour cette intégration.

II.2. TCAO : Concepts et classifications

Dans les années 80, l'utilisation de l'informatique pour gérer la coopération entre membres d'un groupe (travaillant ou pas en même temps, et ou non dans un même lieu) a constitué une révolution concernant le travail coopératif [Cosquer-99]. Cette révolution est à l'origine du domaine intitulé "Travail Coopératif Assisté par Ordinateur", ou TCAO (de l'anglais *Compurter Suported Cooperative Work*, ou CSCW[1]). L'objet du TCAO est en substance d'adapter ces technologies aux besoins des utilisateurs impliqués dans des activités de groupe.

[1] Dans l'article [Schmidt-92], les auteurs nous rappellent que le terme *Computer-Supported Cooperative Work* a été utilisé la première fois en 1984 : Irene Greif et Paul Cashman on désigné ainsi le sujet d'un *workshop* interdisciplinaire organisé sur la thématique de comment offrir un support au travail en groupe avec des ordinateurs.

Les systèmes informatiques, élaborés dans le cadre de la recherche en TCAO, permettant à des groupes d'utilisateurs de collaborer à des buts communs sont qualifiés en anglais de *groupware*. La traduction française couramment utilisée pour ce terme est "collecticiel" [Lévy-90]. En fait, la pluridisciplinarité du TCAO rend difficile la définition de ces systèmes. Une définition assez courante que nous devons à Ellis [Ellis-91] a été traduite par Karsenty [Karsenty-94] de la façon suivante :

"Système informatique qui assiste un groupe de personnes engagées dans une tâche commune (ou but commun) et qui fournit une interface à un environnement partagé".

Collecticiel est également la traduction officielle adoptée par l'Association Française pour la Cybernétique Économique et Technique[1] (ASTI) [ASTI]. Mais très souvent, nous rencontrons également les termes "système collaboratif", "système de TCAO" ou encore "application collaborative" pour désigner le *groupware*. Pour notre part, nous n'emploierons par la suite que les termes "collecticiel" ou "application collaborative" pour désigner ces systèmes informatiques.

Le TCAO est un domaine multidisciplinaire qui concerne, entre autres, les systèmes répartis, l'interface homme-machine, l'automatisation du bureau, *etc*. Cette multidisciplinarité entraîne une grande diversité d'applications qui assistent des groupes d'individus à accomplir des tâches communes. Par conséquent, nous trouvons des divergences importantes parmi les concepts et classifications proposés dans la littérature au sujet des applications collaboratives.

II.2.1. Quelques concepts fondamentaux

Malgré la diversité des applications, nous arrivons à identifier certains concepts fondamentaux que les collecticiels doivent s'efforcer de prendre en compte afin que les tâches coopératives puissent se dérouler correctement. Bien évidemment, pour qu'un collecticiel soit accepté, il faut qu'il offre à ses utilisateurs de bonnes conditions de travail en coopération.

II.2.1.1. La coopération et la collaboration

Les termes "coopération" et "collaboration", et les concepts qu'ils représentent, sont important dans notre présentation. Néanmoins, leur distinction devient rapidement difficile. Dans la littérature ces deux termes sont très souvent utilisés comme synonymes. Les dictionnaires ne nous aident pas vraiment dans cette tâche : une des définition pour "collaborer" étant "coopérer".

Dans les travaux de Ellis [Ellis-94], nous trouvons cependant une distinction entre ces deux termes. Son "modèle des 3C" cherche à caractériser la coopération en trois niveaux distincts, selon l'intensité des relations établies entre les individus et les tâches réalisées : la communication, la coordination, et la collaboration. La communication représente le niveau de coopération le plus lâche, la coordination un niveau intermédiaire, et la collaboration représente le niveau de coopération où les tâches sont les plus imbriquées et fortement liées.

Dans "le modèle du trèfle fonctionnel" [Graham-99] [Lonchamp-03] nous trouvons une classification plutôt fonctionnelle qui n'insiste pas sur une distinction entre

[1] L'ASTI est une société scientifique et professionnelle qui a pour but de donner une visibilité et une cohérence à la communauté des Sciences et Technologies de l'Information.

coopération et collaboration. La notion de coopération est caractérisée par trois axes ou fonctions principales : la communication, la coordination, et la production. La communication correspond à l'échange direct d'informations entre les membres d'un groupe. La coordination consiste à définir les règles d'interaction pour contrôler la contribution des membres du groupe qui participent à un travail commun. La production est caractérisée par le partage d'un espace où les membres peuvent stocker, traiter et partager un ou plusieurs objets communs. Dans [Graham-99], le lecteur trouvera plus de détails concernant les fonctionnalités relatives à chacun de ces trois axes.

Nous prenons en conséquence le parti de considérer les termes de coopération et de collaboration comme étant synonymes, comme l'usage commun et scientifique semblent l'avoir montré.

II.2.1.2. La télé-présence

La télé-présence, ou notion de présence, représente la conscience (de l'anglais *awareness*) commune dans un travail de groupe qui permet de définir le contexte dans lequel le travail se réalise. Cette conscience de groupe correspond à la compréhension que chacun a sur : (i) avec qui il travaille, (ii) l'activité de chacun et (iii) la manière dont les actions de chacun interagissent [Dourish-92].

Dans un collecticiel, à la différence d'une application mono-utilisateur, plusieurs personnes peuvent agir en même temps sur les mêmes artefacts de travail. Dans un tel environnement partagé, les acteurs n'ont pas spontanément connaissance des actions des autres. Ainsi la conscience des activités individuelles et des activités du groupe s'avère une information critique pour rendre possible le succès de la collaboration. Elle peut présenter des avantages tels que : la réduction de l'effort de coordination des actions ; l'anticipation des actions des autres; ou même l'aide à des personnes pour s'intégrer aux activités [Gutwin-02].

Selon l'analyse détaillée de Dix [Dix-97], la notion de télé-présence se décline sous trois formes :

- conscience de la présence des membres du groupe et de leur disponibilité dans le travail coopératif ;
- conscience des actions réalisées par les membres du groupe ;
- conscience des effets consécutifs à ces actions.

Ainsi, la télé-présence, sous ses différentes formes, permet de contrôler les comportements de chaque participant. En effet, pour pouvoir travailler ensemble il est indispensable d'une part d'être conscient de la présence des autres participants et d'autre part d'être au courant de leur travail.

II.2.1.3. La session et les artéfacts de travail

Le concept de session est généralement utilisé en TCAO pour structurer et organiser le travail de groupe. Dans [Haake-99] les auteurs définissent une session par :

- un groupe d'utilisateurs (ou des agents logiciels les représentant) ;
- un espace de travail commun dans lequel les utilisateurs agissent et manipulent des artéfacts de travail ;
- un mode de coopération spécifique exploité par ces utilisateurs.

Ainsi, l'application collaborative doit fournir des opérateurs permettant de gérer une session, tels que créer et détruire une session ; joindre et quitter une session ; et sélectionner ou modifier les modes de coopération, ainsi que les modes de manipulation des artéfact de travail.

La manipulation d'artéfacts évoque la manipulation d'objets physiques que l'on déplace ou modifie sur un bureau réel, comme par exemple déplacer un livre, écrire sur une feuille, utiliser son téléphone, *etc*. Ce concept vise ainsi à déterminer comment les collecticiels peuvent mettre en œuvre le partage d'artefacts de travail (*i.e.* des objet virtuels, des documents, des outils, *etc.*). Le concept de "bureau virtuel" ou d' "espace partagé" (de l'anglais *shared workspace*) correspond à une des notions les plus répandues dans l'organisation d'artefacts de travail tels que des documents et des outils partagés nécessaires aux tâches à être exécutées par le groupe [Dourish-92] [Gutwin-02] [Pinelle-03]. Un bureau virtuel est en fait une instanciation de la métaphore de la pièce, ou *the room metaphor*, décrite en détail dans [Greenberg-02].

Certaines applications collaboratives doivent offrir des fonctionnalités pour contrôler l'accès concurrent aux artefacts de travail partagés. Très souvent ce contrôle est mis en œuvre par un mécanisme de passage de main (connu en anglais sous le terme de *floor control* [Dommel-99]) qui permet à des utilisateurs de prendre la main (ou le *floor*) à distance sur une application partagée.

II.2.2. Classification des applications collaboratives

Il est difficile de définir une classification (ou taxonomie) unifiée des applications de TCAO. En plus de la grande diversité du domaine, chaque classification peut privilégier un aspect particulier. Nous avons ainsi choisi de présenter deux classifications considérées comme incontournables dans la littérature du domaine : l'une concerne des caractéristiques temporelles et spatiales, et l'autre la catégorie fonctionnelle de l'application.

II.2.2.1. La classification espace-temps

L'espace et le temps constituent les dimensions les plus courantes dans les classifications d'applications collaboratives [Ellis-91].

La première dimension, l'espace, concerne la distance géographique séparant les utilisateurs de l'application. Par exemple, les membres d'une réunion peuvent se trouver dans une même pièce ou être carrément situés dans des lieux distants (des bâtiments, des villes, ou bien même des pays différents).

La dimension temporelle caractérise plutôt le type d'interaction entre utilisateurs. Les membres du groupe peuvent interagir en même temps et en direct (les actions d'un participant sont immédiatement transmises aux autres), ce qui définit une collaboration synchrone. Dans ce cas, des problèmes de synchronisation et de gestion de la concurrence se posent. Il faut en effet résoudre les problèmes de conflit induits par exemple lors de modifications simultanées et contradictoires réalisées sur une même donnée. Si une gestion de la concurrence n'est pas mise en place, il est fort probable que des conflits et des incohérences se produiront dans le déroulement de l'application. Les problèmes de contrôle de concurrence et de synchronisation ont été beaucoup étudiés : dans le domaine des systèmes répartis conventionnels et dans le domaine du TCAO (où des problèmes d'ordre humain s'ajoutent aux problèmes techniques) [Munson-96] [Prakash-99] [Yang-04].

Outre la collaboration synchrone, les membres d'un groupe peuvent également agir à des instants différents, c'est-à-dire, les actions sont espacées dans le temps. Il s'agit alors d'une collaboration asynchrone. Dans ce cas, il est important que l'état de l'activité soit conservé en permanence afin que les membres du groupe soient capables d'observer l'historique des actions qui ont été effectuées avant leur arrivée.

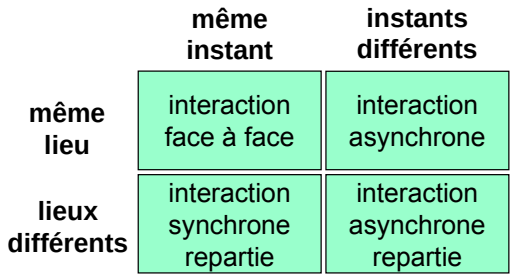

Figure II.1 La matrice espace-temps

La Figure II.1 illustre la "matrice espace-temps", ou encore matrice de Johansen [Johansen-88]. De nombreuses critiques ont été suscitées vis-à-vis de cette classification traditionnelle (de plus en plus les applications tendent à couvrir plusieurs quadrants) et plusieurs propositions ont été développées pour affiner cette classification [Grudin-94].

II.2.2.2. Les catégories fonctionnelles d'applications de TCAO

Beaucoup d'auteurs choisissent d'élaborer une classification par domaines d'application, où une liste de catégories fonctionnelles est définie pour regrouper les applications de TCAO [Bernard-04] [Laurillau-02] [Terzis-99]. Par la suite, nous présentons, à titre d'exemple, une liste non exhaustive de catégories d'applications collaboratives les plus souvent référencées :

- Courrier électronique (courriel) – Cette catégorie de collecticiels représente de loin le moyen de communication asynchrone le plus répandu et le plus utilisé de nos jours. Le courriel désigne le service de transfert de messages envoyés via Internet vers la boîte aux lettres électronique des destinataires choisis par l'émetteur. Une adaptation de ce service de messagerie a connu un grand succès dans le monde de la téléphonie mobile, appelé SMS (de l'anglais *Short Messaging Services*).

- Messagerie instantanée et le forum de discussion – Comme le courrier électronique, ces deux types de systèmes sont dédiés à la communication par échange de messages. Dans le premier cas (également appelé Chat), contrairement au courrier électronique, la communication est conçue pour être instantanée, *i.e.* synchrone. Le deuxième type de système représente des lieux (normalement des sites Web) où les individus peuvent partager leurs connaissances/expériences par des échanges de messages de manière asynchrone. La plupart des forums sont organisés en fils de discussion où un message ou un thème initial détermine un premier fil de discussion. L'ensemble des discussions est généralement visible par les participants, et éventuellement par tous les internautes. Les deux classes principales de forum de discussion sont les *newsgroups* et les *bulletin boards* (ou tableaux d'affichage).

- Gestion de flux de processus – Connue en anglais sous le nom de *workflow management*, cette catégorie représente des systèmes dédiés à la gestion de processus (industriels, commerciaux, administratifs, *etc.*) et à la coordination des différents intervenants au cours d'un processus [Aslst-02] [Ellis-99]. Également dénommé

gestionnaire de procédés, il est souvent défini comme un outil qui prend en charge les documents en cours d'élaboration liés à des procédures et au routage des données. Ces systèmes sont très souvent utilisés par des entreprises dans le but de coordonner les taches exécutées par différents secteurs.

- Système d'aide à la décision – Les GDSS (de l'anglais *Group Decision Support Systems*) sont conçus pour faciliter la prise de décisions en implémentant un sorte de salle de réunion électronique apportant de nombreux outils (par exemple votes, annotation d'idées, *brainstorming*, *etc.*) [Mora-02]. L'anonymat et le droit de parole sont des fonctionnalités normalement mises en oeuvre dans ces systèmes pour encourager les utilisateurs à s'engager dans le processus de prise de décision.

- Outil de partage d'application – Ce type de logiciel permet à plusieurs utilisateurs (travaillant sur des ordinateurs différents) d'utiliser simultanément une application hébergée sur un seul ordinateur (normalement représenté par un serveur) [Ahuja-90] [Begole-99]. Cette fonctionnalité est très souvent implémentée en déportant la fenêtre partagée (voire tout le bureau [Richardson-98]) vers les machines des autres utilisateurs.

- Editeur partagé – Ces systèmes d'édition conjointe (de l'anglais *shared editing* [Prakash-99]) permettent à un groupe d'utilisateurs d'éditer collectivement un document partagé. Ils peuvent être utilisés de façon synchrone ou asynchrone, et ils offrent en général des mécanismes de gestion de versions. Mais la principale complexité de ces systèmes concerne la gestion des tâches concurrentes [Lorcy-00], lorsque des utilisateurs modifient un même document simultanément.

- Tableau blanc partagé – Le système de tableau blanc partagé, comme son nom l'indique, met à disposition une surface de dessin accessible par plusieurs utilisateurs. Il permet ainsi à des utilisateurs de travailler d'une manière synchrone sur des documents 2D. Il est par exemple possible de réaliser des captures d'écran pour les annoter dans le but d'expliquer une idée ou un concept [Kam-05].

- Audioconférence – Les outils d'audioconférence permettent aux utilisateurs de parler à plusieurs. Si d'un coté la qualité de transmission joue un rôle important pour la compréhension de la communication, les flux audio ne consomment pas énormément de bande passante. La communication audio est l'un des moyens de communication le plus riche au niveau signifiant, mais l'utilisation d'un système d'audioconférence à plusieurs comme seul moyen de communication peut entraîner des difficultés dans l'identification des interlocuteurs [Kilgore-03].

- Vidéoconférence[1] – Ces systèmes permettent aux utilisateurs distants de se réunir et communiquer par l'intermédiaire d'un support audio et vidéo en même temps. Si la vidéo peut donner une sensation de présence plus forte que l'audioconférence seule, une vidéoconférence nécessite de disposer d'une bande passante capable de diffuser et recevoir des données audio et vidéo avec une qualité acceptable. Dans [Tang-92] les auteurs suggèrent que, l'audio ayant un rôle plus important que la vidéo pour supporter la collaboration, les systèmes de vidéoconférence ont intérêt à dégrader en priorité la vidéo pour le bénéfice de la qualité de l'audio.

[1] Dans la littérature nous trouvons également le terme "visioconférence", mais selon [Bernard-04] ce dernier désigne une seconde catégorie de systèmes dédiés à des salles de réunion, où plusieurs personnes peuvent communiquer, alors que les systèmes de vidéoconférence sont installés sur des ordinateurs avec des *webcams* individuelles.

- Agenda partagé – Également connu comme calendriers partagés (de l'anglais *group calendars* [Palen-03]), ces systèmes permettent de résoudre des problèmes concernant la planification de tâches. Normalement ces systèmes incluent des fonctionnalités telles que la détection d'incompatibilités dans la planification d'une tâche ou la détermination de plages horaires communes aux membres d'un groupe. Cela permet, par exemple, à l'organisateur d'une réunion d'avoir une vision claire de la disponibilité des acteurs d'un projet.

- Environnement collaboratif virtuel – Les CVEs (de l'anglais *Collaborative Virtual Environments*) représentent une catégorie importante de systèmes de TCAO qui fournissent des facilités de collaboration à travers l'implémentation d'un espace virtuel réparti. L'espace virtuel peut être représenté par de simples environnements textuels, comme dans le cas des environnements *Multi-User Dimension* (MUD) tel que *MOOSE Crossing* [Bruckman-97]. Mais très souvent il s'appuie sur de riches scènes 3D partagées (ce qu'on appelle des mondes virtuels), comme par exemple [Activeworlds] et [Blaxxun]. L'utilisation de la réalité virtuelle est encouragée à cause de sa capacité importante de modélisation et d'interactivité, permettant aux CVEs de résoudre plusieurs problèmes concernant les applications de TCAO : (i) la notion de présence ; (ii) la représentation des métaphores du monde réel (des gestes humains) et des objets (simulations en 3D) ; (iii) la réduction des coûts de transmissions à travers le réseau (notamment par rapport aux systèmes de vidéoconférence).

- Plate-forme collaborative – Cette catégorie spéciale de collecticiels représente en fait des applications qui rassemblent dans un même environnent intégré différents outils associés aux catégories précédentes. Par ailleurs, pour Schmidt et Rodden [Schmidt-96], le caractère dynamique du travail coopératif et ses articulations sont tels que les efforts devraient se tourner vers la mise au point de ces plates-formes fournissant un large éventail de support pour le TCAO : support de partage et d'échange d'information (partage d'application, éditeur partagé, *etc.*) ; support pour la communication (messagerie instantanée, vidéoconférence, *etc.*) ; support pour la prise de décisions, *etc*.

II.3. Développement de systèmes de TCAO

Dans cette section nous allons analyser différentes approches qui ont été employées pour la conception de collecticiels. Nous aborderons d'abord des difficultés d'ordre général pour le développement d'applications collaboratives. Nous passerons ensuite à une description et analyse des approches généralement utilisées dans le développement des collecticiels.

II.3.1. Quelques difficultés

Qu'il s'agisse d'outils pour la communication médiatisée, pour la gestion de contextes coopératifs, de gestionnaires de procédés, *etc.*, l'objectif commun est de fournir un système capable de supporter et de répondre au mieux aux besoins des utilisateurs tout en encourageant la coordination et la gestion des artefacts de travail partagés. Si nous reprenons le modèle du trèfle fonctionnel (voir section II.2.1.1), les trois axes caractérisant la coopération impliquent qu'une application collaborative doit supporter à la fois des mécanismes pour la communication, pour la coordination et pour la production collaborative.

Il ne faut pas oublier qu'il s'agit de supporter des interactions dirigées par des utilisateurs. Ainsi, les collecticiels doivent prendre également en compte des aspects plus "humains" de la coopération tels que la gestion d'un groupe de personnes et la télé-présence. De plus, différentes équipes et organisations exécutant différentes tâches, demandent autant de fonctionnalités différentes [Haake-99].

Compte tenu de tous ces aspects et contraintes à prendre en compte, c'est bien naturel que les collecticiels rencontrent toujours des difficultés à atteindre une acceptation importante de la part de vrais groupes de travail [Palen-03]. Nous retrouvons en fait, à l'origine de cet échec, la façon dont les personnes travaillent généralement en groupe, qui, comme cela est signalé dans [Hummes-00], n'est pas tout à fait prévisible. Par conséquent, il est pratiquement impossible pour les concepteurs/développeurs de collecticiels de prédéfinir par anticipation les besoins réels des utilisateurs vis-à-vis des artefacts de collaboration nécessaires au support du travail en groupe.

Ainsi, selon Hummes, "un système collaboratif doit permettre la création et l'insertion de nouveaux modules et artefacts coopératifs". Il faut donc que les développeurs conçoivent les applications collaboratives tout en prenant en compte les problèmes liés à l'intégration des nouvelles fonctionnalités qui n'ont pas été prévues à l'avance, mais qui s'avèrent nécessaires pendant le déroulement du travail. En vérité, depuis longtemps plusieurs auteurs (par exemple [Edwards-96], [Malone-95], [Roseman-93], [Shen-02] et [Wang-00]) définissent cette capacité de permettre au système de s'adapter à des besoins particuliers, caractéristique couramment appelée flexibilité, et considérée comme indispensable en TCAO. Effectivement, étant donné que les activités collaboratives impliquent plusieurs personnes qui expriment souvent des besoins de travail en groupe différents et parfois imprévisibles, il est fort probable que des systèmes plus flexibles soient mieux acceptés par les utilisateurs [Kahler-00].

II.3.1.1. La flexibilité, l'extensibilité et la malléabilité

Nous trouvons souvent deux autres propriétés associées à la flexibilité : (i) l'extensibilité, qui selon une définition générale, privilégie l'ajout de nouveaux comportements aux systèmes, soit par l'intégration de nouveaux modules ou composants, soit par l'extension des modules/composants existants dans l'application [terHofte-98] ; et (ii) le *tailoring* [Dourish-96], traduit en français par "malléabilité" [Bourguin-00], qui définit des notions d'adaptabilité et de flexibilité pour les utilisateurs finaux (ou à un responsable plus expérimenté, tel qu'un administrateur).

Mørch [Mørch-97] distingue trois formes de malléabilité : le paramétrage, l'intégration et l'extension de l'application. Le paramétrage permet de sélectionner une ou plusieurs valeurs pour certains paramètres d'une application afin de la spécialiser. Quant à la distinction entre intégration et extension, Mørch insiste que l'extensibilité est associée à la possibilité d'étendre une application en modifiant des parties spécifiques de cette application, pour qu'elle puisse répondre à des besoins provenant de l'utilisateur ; l'intégration, quant à elle, se limite à l'ajout et à la composition de nouveaux modules ou composants à l'application.

Nous avons décidé d'emprunter la notion d'extensibilité dans le sens plus général du terme (telle qu'elle est présentée dans [terHofte-98]). Ainsi, une application peut être également étendue par le biais de l'intégration de nouvelles fonctionnalités sous la forme de nouveaux composants, modules ou d'autres applications.

II.3.2. Les approches de développement

Tout en essayant de prendre en compte ces différents aspects, conséquences de la diversité intrinsèque au domaine du TCAO, différentes approches ont été utilisées dans le développement d'applications collaboratives. Nous allons par la suite décrire ces approches tout en identifiant comment elles peuvent influencer l'acceptation des applications (développées selon ces approches) au sein de véritables groupes de travail.

II.3.2.1. Développement *from scratch*

Le développement *from scratch*[1] concerne la conception complète de nouveaux systèmes de TCAO "à partir de rien", c'est-à-dire, sans faire appel à des composants ou des sous-systèmes implémentant des fonctionnalités générales ou spécifiques. Le principal avantage de cette approche est que le développeur est libre de prendre ses propres décisions concernant la spécification, la modélisation et l'implémentation de l'application.

Puisque ces systèmes représentent souvent des solutions propriétaires ne s'appuyant pas sur des modèles de conception ouverts, ils risquent fort de présenter les inconvénients classiques des applications fermées : non-standardisation, manque de flexibilité et d'interopérabilité. En outre, le fait de développer tout un système sans faire appel à des solutions pré-existantes peut conduire à "ré-inventer" la roue, sans parler de l'augmentation des coûts et de la fiabilité, probablement inférieure, du système. Finalement, puisque toutes les fonctionnalités collaboratives ont été statiquement imbriquées dans l'application, cela peut représenter une vraie surcharge architecturale étant donné que certaines fonctionnalités peuvent même ne pas être utilisées.

Il s'agit d'une approche en général utilisée pour le développement d'applications qui ne présentent pas trop de complexité vis-à-vis du nombre de fonctionnalités collaboratives offertes (par exemple *CoLab* [Hoyos-05b], *Skype* [Skype], *Shared Calendar* [SharedCalen], *etc.*). Or, nous trouvons également des plates-formes collaboratives, surtout dans le domaine commercial, qui ont été développées selon cette approche (par exemple *KnowEdge eLearning Suite* [KnowEdge], *Groove Virtual Office* [Groove], *Avilar WebMentor* [WebMentor], *etc.*).

II.3.2.2. Développement s'appuyant sur des boîtes à outils

Une boîte à outils (de l'anglais *toolkit*) est une bibliothèque de composants logiciels en charge de gérer des structures de données et de réaliser des opérations communes pour le développement d'applications. Ces composants sont en général accessibles par un programme via des appels procéduraux. L'objectif d'une boîte à outils est ainsi de fournir un cadre de programmation adéquat pour implémenter certaines parties d'une l'application tout en réduisant l'effort de programmation.

Nous nous intéressons aux boîtes à outils dédiées au développement de collecticiels (*groupware toolkit* ou *CSCW toolkit*). Il s'agit d'environnements qui fournissent un ensemble d'éléments (outils) implémentant des services spécialisés généralement requis pour la coopération.

L'approche de développement s'appuyant sur des boîtes à outils, comparée à l'approche de développement *from scratch*, présente l'avantage de réduire le coût de

[1] Egalement appelé "développement *ad hoc*".

développement et de faciliter le prototypage rapide de nouveaux collecticiels. Par ailleurs, l'utilisation d'une boîte à outils permet de concevoir des collecticiels variés et adaptés aux besoins des utilisateurs, tout en gardant un environnement homogène (notamment en terme d'interface graphique), ce qui peut faciliter leur utilisation [Laurillau-02].

Une propriété importante de ces types de systèmes est le bas niveau d'extensibilité qu'ils peuvent offrir aux développeurs. Dans le cas de boîtes à outils plus spécialisées, comme *SDGToolkit* [Tse-04], qui est un *toolkit* dédié au SDG (*Single Display Groupware*), ou *Collabrary* [Boyle-02], qui représente un *toolkit* pour le prototypage de collecticiels pour la manipulation de données multimédia, cette propriété peut ne pas être essentielle. Cependant, dans le cas de boîtes à outils à vocation plus générale, un manque d'extensibilité constitue un grave inconvénient. Par conséquent, pour pouvoir offrir plus de possibilités aux développeurs, ces boîtes à outils peuvent devenir très volumineuses, impliquant une prise en main par le développeur de plus en plus difficile. Par ailleurs, comme dans le cas du développement *from scratch*, il est impossible d'avoir des boîtes à outils offrant toutes les fonctionnalités possibles (notamment celles qui n'ont pas été prévues !).

L'extensibilité consiste donc à permettre aux développeurs de créer de nouveaux outils (implémentant de nouveaux comportements) et de les incorporer à la boîte à outils. Les boîtes à outils offrant une "implémentation ouverte" (de l'anglais *Open Implementation*, une approche décrite par Kiczales [Kiczales-96]) et celles que sont carrément *open source*[1], sont extensibles de manière intrinsèque (par exemple *Prospero* [Dourish-98]). Mais le niveau d'expertise exigé des développeurs peut être décourageant étant donné qu'ils doivent bien comprendre les stratégies d'implémentation employées à l'intérieur du système.

Pour aider le développeur dans cette tâche, certains systèmes offrent quelques facilités d'extensibilité. *COAST* [Schuckmann-96], par exemple, caractérise une boîte à outils qui, s'appuyant sur les principes de l'orientation objet, offre un ensemble de classes abstraites[2] et génériques qui peuvent être spécialisées par le développeur afin de créer de nouveaux éléments. Mais ceci implique des efforts de développement et d'implémentation qui doivent être entrepris en dehors du contexte de la boîte à outils (c'est-à-dire, en utilisant un environnement de programmation extérieur) afin de concevoir de nouveaux outils. Ces nouveaux outils resteront de plus spécifiques à la boîte à outils.

D'autres solutions ont été proposées pour assurer flexibilité et extensibilité [Dourish-00] [Roseman-96] [Shen-04]. Nous avons par exemple *GroupKit* [Roseman-96], qui est certainement la boîte à outils la plus connue dans le monde du TCAO. Il s'agit d'un environnement pour le développement de collecticiels développé en langage interprété (en l'occurrence Tcl/TK). Outre l'ensemble des services pour la coopération présenté sous la forme de *groupware widgets*[3], *GroupKit* fournit un outil appelé *Class Builder* qui permet aux développeurs de créer de nouveaux *widgets* ou d'adapter le comportement des *widgets* existants.

Pendant un moment l'approche de développement basée sur des boîtes à outils a été très répandue. Néanmoins les développeurs ont vite constaté que ces boîtes à outils (extensibles ou non) présentent une contrainte de taille : elle n'offrent pas de mécanismes

[1] Terme suggéré par *Christine Peterson* du *Foresight Institute* [Foresight] pour caractériser les logiciels dont le code source est disponible, modifiable et "redistribuable" sous certaines conditions.
[2] En programmation orientée objet, une classe abstraite est une classe "non-instanciable" qui contient des fonctions virtuelles sans code (aussi dites abstraites). Elle sert juste de base à d'autres classes héritées.
[3] Un *widget* représente un élément graphique d'interface.

permettant la réutilisation ou l'intégration d'outils réalisés par des tiers. Tout nouvel outil doit être conçu dans le cadre de développement, souvent en utilisant une application propriétaire, définie par la boîte à outils. Et les nouveaux outils ne sont utilisables que dans le cadre de la boîte à outils pour laquelle ils ont été conçus.

II.3.2.3. Développement s'appuyant sur des *frameworks* et des plates-formes

De nombreux projets de recherche se fixent comme objectif de construire des *frameworks* ou des plates-formes pour le développement d'applications collaboratives. Il s'agit en général de solutions plus génériques que les boîtes à outils qui cherchent notamment à supporter des propriétés telles que la flexibilité et l'extensibilité.

Les *frameworks*, couramment appelés en français "architectures cadre" ou "cadre d'application", fournissent une structure réutilisable qui spécifie les fondements de l'application. Normalement, cette structure se présente sous la forme d'une architecture générique ou conceptuelle. Les *frameworks* peuvent également être associés à la notion de *pattern* (ou "patron") qui décrivent des solutions reconnues pour une classe de problèmes (par exemple [Tarpin-Bernard-98]).

Une plate-forme offre en général un support générique d'exécution et de développement de collecticiels en définissant une architecture et des briques logicielles implémentant des mécanismes de contrôle pour la collaboration. Certaines plates-formes sont développées en s'appuyant sur des *frameworks* (par exemple, [Barthelmess-05] et [Domingos-97]). Néanmoins, une distinction consensuelle entre *frameworks* et plates-formes s'avère difficile, surtout quand nous trouvons le terme *framework* désignant des environnements fournissant un ensemble de bibliothèques ou briques logicielles pour permettre le développement d'applications (par exemple [.NET]). Nous allons ainsi employer ces termes selon les critères choisis par chaque auteur des travaux cités par la suite.

Un *framework* ou une plate-forme peut être définie pour supporter la construction de nouvelles applications ou la transformation d'applications existantes en applications collaboratives. Dans cette dernière catégorie, nous avons par exemple la plate-forme *XTV* [Abdel-Wahab-91] et les *frameworks Habanero* [Jackson-99] et *DISCIPLE* [Li-99]. Ces systèmes permettent de partager entre plusieurs utilisateurs des applications originalement conçues pour un seul utilisateur. En particulier, *Habanero* et *DISCIPLE* offrent aux développeurs les outils nécessaires à la création de collecticiels implémentés en *Java* par l'intégration et le partage de différentes applications mono-utilisateur (l'intégration d'applications originalement collaboratives est également envisageable). Malgré cette flexibilité concernant la possibilité de combiner différentes applications dans un même environnement, il s'agit beaucoup plus d'une "cohabitation" que d'une réelle intégration, car il n'existe pas vraiment d'interactions entre ces applications.

Dans la catégorie des *frameworks* et des plates-formes pour la construction de nouvelles applications collaboratives, une tendance actuelle des concepteurs de ces systèmes consiste à réaliser un développement modulaire en s'appuyant sur des composants. En effet, au cours de ces dernières années dans le domaine de l'ingénierie de systèmes, plusieurs travaux ont porté sur les approches orientées composants. Le développement à base de composants logiciels est devenu un facteur majeur pour rendre possible la composabilité, l'extensibilité et la malléabilité [Slagter-00].

Il existe plusieurs définitions du terme "composant logiciel". En particulier la définition de Szyperski [Szyperski-02] est très souvent adoptée, notamment dans le

domaine du TCAO. Szyperski définit un composant logiciel comme : "une unité de composition avec des interfaces contractuellement spécifiées et des dépendances explicites sur son contexte. Un composant logiciel peut être déployé indépendamment et il est sujet à des compositions par des tiers".

Différents auteurs considèrent que les collecticiels de demain seront davantage des assemblages de composants (par exemple [Farias-00b] et [Grundy-02]). Ils partent du principe que la plupart des fonctionnalités requises existeront déjà sous forme de composants développés par des tiers. En s'appuyant sur des assemblages de composants, les systèmes peuvent définir des environnements collaboratifs complètement intégrés.

Ainsi, au contraire des boîtes à outils, un schéma d'interaction entre composants doit être défini afin de permettre l'intégration de différents composants dans une même plate-forme. *TeamWave Workplace* (*TeamRooms* à l'origine) [Roseman-97] par exemple propose une extension basée composants de *GroupKit*. Une architecture et un modèle de composant spécifique sont définis de manière simple afin d'encourager le développement de nouveaux composants par des tiers. *Neem* [Barthelmess-05] représente également une plate-forme définissant une infrastructure de composants réutilisables et extensibles. Mais dans les deux cas, la réutilisation des composants est limitée au système en question. Ainsi, le développeur est contraint d'utiliser les composants spécifiques au système.

Afin de rendre possible l'intégration de composants développés sans avoir de système spécifique cible, la construction de collecticiels converge de plus en plus vers l'utilisation d'intergiciels (de l'anglais *middleware* [Geihs-01]) définissant des modèles généraux de composants). Des exemples répandus de modèles de composants sont *CCM* (*CORBA Component Model* [OMG-04]), *EJB* (*Entreprise JavaBeans* [EJB]) et *.NET* [.NET]. Ces modèles de composants offrent une séparation entre les fonctionnalités métier du composant et les fonctionnalités système (sécurité, répartition, cycle de vie, *etc.*). Cette séparation est réalisée suivant le paradigme composant/conteneur [Szyperski-02].

Ainsi, de nombreux systèmes pour le développement d'applications collaboratives ont été développés en s'appuyant sur ces modèles généraux de composants, notamment ceux définis par *CORBA* et *JavaBeans* [Slagter-00]. Nous avons par exemple les systèmes basés sur *CORBA*, tels que *CoCoWare* [Slagter-01], *DARE* [Bourguin-00] et la plate-forme décrite en [terHofte-96], et sur *JavaBeans*, tels que *ACOST* [Hummes-00], *EVOLVE* [Stiemerling-00], *MAUI toolkit* [Hill-04] et *ANTS* [Garcia-01].

L'intérêt de ces systèmes est ainsi d'assurer la propriété d'extensibilité par le biais de l'intégration de composants développés par des tiers[1] qui n'ont pas été *a priori* préconisés pour ces systèmes. La condition à respecter est que ces composants s'appuient sur le même modèle général de composants. Certains systèmes offrent des possibilités d'extensibilité encore plus élaborées permettant que de nouveaux composants soient intégrés aux collecticiels en temps d'exécution (par exemple [Hummes-00]). En outre, puisque les composants originellement disponibles dans ces systèmes (généralement des composants intégrant les mécanismes de base nécessaires à la coopération) ont été développés selon le modèle général de composant utilisé par le système, ils peuvent être réutilisés par des tiers, ou même intégrés dans un environnement de développement intégré (EDI[2]) standard. Par exemple, les *MAUI components* (des composants offrant des

[1] Par exemple, des composants disponibles sur le marché, aussi appelés composants *COTS* (abréviation anglo-saxonne de "commercial off the shelf").
[2] Ou *IDE* de l'anglais *Integrated Development Environment*.

fonctionnalités multi-utilisateurs fournis par *MAUI toolkit*[1]) peuvent être utilisés dans des EDIs *Java* tels que *NetBeans* [NetBeans] et *JBuilder* [JBuilder].

Ainsi, l'approche de développement s'appuyant sur des modèles généraux de composants définis par des intergiciels facilite l'intégration d'autres fonctionnalités coopératives sous la forme de composants logiciels. L'utilisation de telles technologies impose toutefois des contraintes vis-à-vis de l'environnement d'exécution. Autrement dit, cela reste difficile (voire impossible) d'intégrer aux collecticiels des composants, ou carrément d'autres applications collaboratives, qui n'ont pas été conçus en se reposant sur ces intergiciels.

Même si les intergiciels viennent à l'aide du développeur en lui cachant plusieurs problèmes liés à la conception d'applications collaboratives, comme par exemple la distribution géographique des participants, ils deviennent de plus en plus complexes [Kordon-05]. Cette complexité est souvent liée au fait que les concepteurs des intergiciels cherchent à établir des systèmes répartis toujours plus sophistiqués [Colyer-03]. Cette tendance amène les développeurs à être confrontés au même ensemble de problèmes que les intergiciels étaient censés résoudre – les systèmes pour le développement de collecticiels, ainsi que les collecticiels développés, risquent de devenir trop complexes. Dans le paragraphe II.4.2.1 nous allons aborder plus en détail quelques intergiciels en tant que solutions pour l'intégration d'applications.

II.4. Vers l'intégration d'applications collaboratives existantes

II.4.1. Motivations

Indépendamment de l'approche de développement choisie, la conception de nouvelles applications collaboratives qui soient extensibles se présente comme un grand défi pour les développeurs. Si d'un coté l'utilisation de boîtes à outils s'avère simple et rapide, le degré d'extensibilité obtenu est très restreint. Si les développeurs choisissent de s'appuyer sur des *frameworks* ou des plates-formes, la possibilité d'intégrer de nouveaux composants est également contraignante puisque ces derniers doivent suivre un schéma d'interaction imposé par le système en question. Ou alors les développeurs doivent faire face à la complexité des intergiciels afin de pouvoir intégrer des composants disponibles sur le marché.

Dans la pratique, nous trouvons sur le marché soit des collecticiels se focalisant sur des aspects spécifiques de la coopération, et qui ont souvent été développés selon l'approche *from scratch*, soit des plates-formes collaboratives, qui ont souvent été développées selon des *frameworks* très sophistiqués [Eseryel-02]. Dans cette deuxième catégorie, nous trouvons des systèmes commerciaux très diffusés tels que *Microsoft Live Meeting* [LiveMeeting] et *IBM Lotus Software* [Lotus]. Ces deux plates-formes offrent tout un éventail d'outils pour la communication, la gestion et le partage de documents, des calendriers partagés, *etc*. La possibilité d'intégration est effectivement considérée dans *Live Meeting* qui peut facilement interagir avec d'autres outils *Microsoft* (notamment les outils de *Microsoft Office*). *Lotus Software*, de son coté, assure l'extensibilité par son application *Domino*, qui offre une *suite* d'outils de développement permettant aux

[1] Les auteurs désignent *MAUI* comme une boîte à outils, mais en réalité, il surpasse ce concept puisque les outils représentent des vrais composants logiciels.

développeurs de créer de nouvelles fonctionnalités et de les ajouter au collecticiel. Mais malgré toutes les possibilités d'extensibilité offertes par ces plates-formes, elles sont loin d'être considérées comme une panacée. Outre le coût associé à ces systèmes, l'utilisateur se trouve toujours limité à une plate-forme donnée. Autrement dit, il est confronté à un choix limité d'outils de collaboration.

Ainsi, des utilisateurs finissent par composer leurs propres "environnements collaboratifs" en choisissant, selon leurs besoins, différentes applications offrant des fonctionnalités spécifiques, en les exécutant en même temps mais indépendamment les unes des autres. Néanmoins, compte tenu du fait que ces applications collaboratives sont en réalité employées pour réaliser une même tâche ou un ensemble commun de tâches pour le travail en groupe, il devrait exister une liaison, voire des interactions entres ces différentes applications. En effet, l'intégration de ces applications collaboratives pourrait être très bénéfique aux utilisateurs : différentes fonctionnalités pourraient être combinées et contrôlées de manière dynamique, améliorant ainsi la flexibilité du système. Par ailleurs, par l'intégration d'applications fournissant des technologies complémentaires, la valeur de la technologie dans son ensemble est supérieure à la somme des valeurs associées à chaque application [Mangematin-96]. Il y a une vingtaines d'années, Wilson [Wilson-88] dans une révision des travaux réalisés dans le domaine du TCAO montrait déjà que le besoin d'environnements collaboratifs intégrés était évident.

Teege [Teege-96] récapitule les deux démarches à suivre pour construire de tels environnements collaboratifs intégrés. La première démarche est celle que nous avons abordée jusqu'à présent ; elle consiste à les concevoir et les développer dès le début en tant que systèmes intégrés. Leur mise en œuvre peut être réalisée selon une des approches de développement décrites dans le paragraphe II.3.2. La deuxième démarche, au contraire de ces approches de développement, ne cherche pas à concevoir de nouveaux collecticiels. En fait l'objectif est de construire des plates-formes ou environnements collaboratifs intégrés en rassemblant des collecticiels existants tout en assurant leur interopérabilité. Dans les paragraphes suivants, nous allons tout d'abord étudier la problématique associée à l'intégration d'applications de façon générale, pour ensuite aborder les démarches spécifiques à l'intégration d'applications collaboratives.

II.4.2. Plates-formes et environnements généraux d'intégration

La problématique associée à l'intégration d'applications ne se restreint pas au domaine du TCAO. Dans de nombreux domaines, et notamment dans le domaine du développement d'applications réparties, l'un des principaux axes de recherche concerne la possibilité de faire interopérer différents systèmes. L'interopérabilité ou l'intégration[1] entre les applications facilite la coopération au sein d'une organisation ou entre différentes organisations sans les contraindre à utiliser exactement les mêmes systèmes ni à posséder les mêmes équipements.

La notion d'intergiciel a fait son apparition afin de faciliter le développement d'applications réparties et de gérer les intéractions entre ces applications via des plates-formes hétérogènes (par exemple diversité du matériel, des systèmes d'exploitation ou des langages de programmation). Un intergiciel représent ainsi une solution architecturale au

[1] Nous trouvons dans la littérature le terme d'"interopérabilité" distingué du terme d'"intégration", mais également employé de façon intercalée [Brownsword-04]. Sauf mention explicite, nous avons choisi de ne pas faire de distinction.

problème de l'intégration de différentes applications tout en imposant à ces applications une interface de service commune.

Également dans le cas des entreprises, la nécessité de collaborer efficacement via l'intégration des applications de l'entreprise n'a jamais été aussi importante. L'environnement économique actuel impose de nouvelles stratégies, telles que le partenariat, incitant de plus en plus les entreprises à échanger des données et à interagir les unes avec les autres. Dans ce contexte, le concept de EAI (*Enterprise Application Integration*) est apparu pour désigner l'ensemble des méthodes, technologies et outils destinés à l'intégration des différentes composantes des systèmes d'information des entreprises.

Parallèlement à l'évolution des intergiciels et de l'EAI, dans le domaine de l'Internet, les technologies Web ont émergé et remporté un succès important, en permettant des interactions simples avec des ordinateurs à distance. Plus récemment, les services Web, qui reprennent la plupart des principes du Web en les appliquant à des interactions ordinateur-ordinateur, se présentent comme une technologie potentielle pour l'intégration d'applications à travers l'Internet.

II.4.2.1. Les intergiciels conventionnels

Les solutions à base d'intergiciels sont largement utilisées dans les architectures des systèmes d'information [Schantz-02]. Elles peuvent être divisées en deux classes principales : les intergiciels orientés objet et les intergiciels orientés messages. Dans la première classe, très souvent, les intergiciels fournissent un modèle de composants. La communication entre objets (ou composants) est normalement synchrone et s'appuie sur le mécanisme *RPC* (*Remote Procedure Call*). Les intergiciels orientés messages, ou MOM (*Message-Oriented Middeleware*), offrent une interopérabilité dont le paradigme d'interaction est asynchrone et s'appuie sur l'échange de messages. Néanmoins, nous remarquons que les trois grands intergiciels qui ont remporté un succès certain dans ce domaine suivent en fait les principes de l'orientation objet : *CORBA*, *DCOM* et *Java RMI*.

a) CORBA

L'*OMG* (*Object Management Group* [OMG]) est un organisme international qui s'est fixé pour objectif de définir des standards pour la mise en place et l'intégration d'applications réparties. Son premier travail fut la définition de l'*OMA* (*Object Management Architecture* [OMA]) spécifiant un modèle d'objet et une architecture d'intégration. La spécification de *CORBA* (*Common Object Request Broker* [OMG-04]) suit directement la définition de l'*OMA*. Nous trouvons le concept d'*ORB* (*Object Request Broker*) au centre de l'architecture *CORBA*. Il s'agit d'un mécanisme qui permet, de façon transparente, de localiser des objets et d'établir des relations de type client/serveur afin d'invoquer des méthodes de ces objets. Il existe aujourd'hui plusieurs implémentations d'*ORB*s (par exemple [Orbix] et [Visibroker]).

L'interopérabilité entre objets hétérogènes est rendue transparente par la spécification d'interfaces en utilisant le langage *CORBA-IDL* (*CORBA Interface Description Language*). Ce langage permet de spécifier l'interface de chaque objet en décrivant toutes ses méthodes, et leurs paramètres, qui peuvent être invoquées à distance. Ainsi le service fourni par un objet *CORBA* est décrit indépendamment de son implémentation. Un compilateur génère, à partir d'une spécification *CORBA-IDL*, des souches (*stubs*) qui sont destinées aux clients, et des squelettes (*skeletons*) qui sont destinés

aux serveurs, et ce dans différents langages d'implémentation (*C*, *C++*, *Java*, *etc*.). Ces objets mis en œuvre sur des systèmes hétérogènes peuvent ensuite interopérer.

Le principe de fonctionnement d'un *ORB* consiste à intercepter la requête du client, trouver un serveur, lui passer les paramètres et invoquer la méthode correspondante à l'appel pour ensuite retourner les résultats au client. En général, l'appel est fait de façon statique, correspondant à un appel bloquant de type *RPC*. Néanmoins, *CORBA* possède un MOM spécifique permettant également de réaliser des appels dynamiques (au moyen d'échange de messages) à des objets dont les interfaces ne sont pas connues *a priori*. Un réceptacle d'interfaces contenu dans l'*ORB* est utilisé pour permettre aux objets de découvrir dynamiquement les autres objets.

En spécifiant comment différents *ORB*s peuvent interopérer, *CORBA* représente la réponse de l'*OMG* au problème de l'interopérabilité entre applications. Le protocole *GIOP* (*General Inter-ORB Protocol*), qui permet de coder les messages associés à la sémantique des échanges requête-réponse, prévoit un support d'interopérabilité entre *ORB*s. Le mécanisme *IIOP* (*Internet Inter-ORB Protocol*) de ce protocole définit comment la communication entre différents MOMs basée sur TCP/IP doit s'effectuer.

b) DCOM

DCOM [Microsoft-96] (*Distributed Component Object Model*) est un intergiciel propriétaire développé par *Microsoft*. Il s'agit d'une extension distribuée de *COM* [COM], une architecture logicielle à base de composants. Cette dernière technologie s'appuie sur *OLE* (*Object Linking and Embeding*) pour permettre l'interaction entre différentes applications *Windows*.

COM définit une spécification et l'implémentation correspondante, et selon cette spécification, des composants (dits objets *COM*) peuvent fournir des services. Grâce à un ensemble d'interfaces spécifiées dans le langage *MIDL* (*Microsoft Interface Description Language*), les objets *COM* sont exposés et leurs services peuvent alors être invoqués par des clients. En utilisant ce langage, comme dans le cas de *CORBA*, la description du service reste découplée de son implémentation. En fait, le codage des objets *COM* et des clients peut être réalisé dans n'importe quel langage supportant le format binaire de *COM*.

La principale différence entre *COM* et son extension distribuée est que le premier assure l'interopérabilité des applications à l'intérieur d'un même ordinateur, tandis que *DCOM* permet de faire interopérer des applications sur des plates-formes *Windows* distinctes à travers un réseau. Par conséquent, *DCOM* doit faire face à toutes les contraintes imposées par *COM*. La contrainte la plus importante concerne le fait que l'environnement doit être homogène, c'est-à-dire, *DCOM* finit par être une architecture distribuée fermée. L'interopérabilité n'est alors supportée qu'entre plates-formes *Windows*.

c) Java RMI

Sun Microsystems a développé, en tant que partie intégrante de *Java,* la plate-forme de communication *RMI* [*RMI*] (*Remote Method Invocation*). *RMI* fournit les fonctionnalités nécessaires à l'appel à distance. Il met en œuvre un mécanisme de communication très proche de celui du *RPC*, mais au lieu d'invoquer des procédures, il invoque des méthodes. *Java RMI* permet ainsi d'effectuer des appels de méthodes sur des objets locaux et des objets distants en utilisant le même mécanisme. Une référence sur un objet distant peut être obtenue par l'intermédiaire d'un serveur de nom, où sont enregistrés les objets pouvant être appelés.

Pour qu'un objet puisse être reparti, il doit être une instance d'une classe mettant en œuvre l'interface *Remote*. Outre la compilation habituelle *Java*, cette classe doit subir une compilation spécifique afin de produire deux classes mettant en œuvre le processus de communication distribuée : (i) la souche (*stub*), utilisée du coté client pour invoquer les méthodes à distance, et (ii) le squelette (*skeleton*) associé à l'objet pour décoder les appels reçus des clients.

Java RMI fut à l'origine développé pour ne faire communiquer entre eux que des objets *Java*. Néanmoins, pour ne pas restreindre l'interopérabilité à des applications *Java*, Sun a étendu *Java RMI* de telle sorte qu'il utilise *IIOP* comme protocole de communication. Par conséquent, des objets *Java* peuvent également interopérer avec des objets *CORBA* (pouvant être codé en différents langages).

II.4.2.2. *Enterprise Application Integration*

L'EAI est un concept qui est la conséquence du besoin d'intégrer des systèmes d'information pour former un tout cohérent et opérationnel à la fois au sein des entreprises et entre entreprises (ce deuxième cas est également appelé *inter-EAI*) [Alonso-04a]. Intergiciel et EAI ne sont pas des concepts tout à fait orthogonaux. EAI peut être considéré comme un pas en avant dans l'évolution des intergiciels. Si les intergiciels ont été également conçus pour faciliter le développement de nouvelles applications réparties, EAI cherche à étendre les capacités des intergiciels afin de faciliter l'intégration.

Les solutions d'EAI reposent généralement sur l'utilisation d'un bus de communication en lieu de communications directes entre applications. Autrement dit, nous avons une infrastructure d'intégration dédiée qui gère la communication et les échanges des applications. La plupart des applications peuvent ainsi se brancher sur cette infrastructure par l'intermédiaire d'interfaces génériques (également appelés "connecteurs").

Une des couches techniques d'une solution EAI est l'infrastructure de communication, très souvent implantée en utilisant des MOMs. Ce choix s'explique par le fait que les MOMs favorisent le découplage entre applications. Elles ne communiquent pas directement entre elles, mais s'échangent des messages par l'intermédiaire d'un médiateur. Le découplage des applications est un moyen de garantir qu'il n'y ait pas de conséquence au niveau du fonctionnement du système, lors de l'ajout ou du retrait d'une application (les applications sont en quelque sorte autonomes). Les MOMs ont déjà fait leurs preuves dans le contexte EAI depuis quelques temps. Par exemple, depuis plus de quinze ans, la solution d'EAI *WebSphere MQ* [WMQ] d'*IBM* (initialement connue comme *MQSeries*) s'appuie sur un MOM.

Les solutions d'EAI se sont longtemps appuyées sur des infrastructures propriétaires (par exemple *TIBCO BusinessWorks* [BusinessWorks] et *BEA WebLogic Integration* [WebLogic]). Mais depuis la spécification de standards dans la plate-forme *J2EE* (*Java 2 Enterprise Edition* [J2EE]), telle que la définition de *JMS* (*Java Message Service* [JMS], qui définit une interface d'accès standard en *Java* pour les MOMs) par *Sun* et ses partenaires, il y a eu une véritable avancée dans le monde de l'EAI. Les solutions d'EAI adoptent progressivement des MOMs conformes à JMS. S'appuyant pour la plupart sur le langage *Java*, et notamment sur *JMS*, des solutions d'EAI *open source* ont également fait leur apparition : des plates-formes telle que *JOnAS* [JOnAS], une implantation *open source* des spécifications *J2EE*, ou des solutions qui ne représentent pas vraiment de plates-formes prêtes à l'emploi, mais qui peuvent être utilisées en tant que briques

techniques pour l'implantation de solutions d'EAI (par exemple *OpenAdaptor* [OpenAdaptor] et *OpenEAI* [OpenEAI]).

II.4.2.3. Les services Web

Le terme "services Web" (de l'anglais *Web services*) est très souvent utilisé à l'heure actuelle et il présente différentes définitions dans la littérature. Une définition plus précise est fournie par *W3C* (*World Wide Web Consorcium*) [W3C], qui accompagné de l'*OSI* (*Open Source Initiative*) [OSI], représente le comité de coordination responsable de l'architecture et de la standardisation des services Web. La définition du W3C est la suivante : "*A Web Service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts, and which supports direct interactions with other software applications using XML-based messages via Internet-based protocols.*" [W3C-04].

Plus informellement, nous pouvons dire que les services Web sont des applications qui définissent un ensemble d'interfaces, s'appuyant sur *XML* (*eXtensible Markup Language* [XML]), et qui peuvent interagir dynamiquement entre elles et avec d'autres applications grâce à l'échange de messages basés sur *XML* utilisant les protocoles de transport Internet disponibles.

Ainsi, comme pour le *World Wide Web*, l'un des principaux facteurs de succès des services Web réside dans l'utilisation d'une pile simple de protocoles s'appuyant sur des normes ouvertes. De plus, les services Web partagent une architecture commune qui a pour fondement le concept d'architecture orientée service (*Service-Oriented Architecture* ou *SOA* [Perrey-03]). Il s'agit de répartir une application en petites unités fonctionnelles ou services. Une architecture orientée service doit se concentrer sur la façon dont ces services sont décrits et organisés pour supporter leur découverte, leur utilisation et leur dynamique.

Le fonctionnement de base des services Web dépend de trois technologies qui reposent sur le langage *XML* :

- *SOAP* (*Simple Object Access Protocol* [W3C-03]) – Les services Web communiquent par messages. Ils supposent que *SOAP* est employé dans les couches basses de la pile de protocoles en isolant le transfert des messages des détails de la couche transport. *SOAP* est un protocole qui définit la structure des messages échangés par les applications via Internet, chaque message étant un document *XML*.
- *WSDL* (*Web Service Description Language* [W3C-01]) – *WSDL* fournit un mode de description de l'interface d'un service Web. Il permet de décrire un contrat de service qui fourni des informations telles que les méthodes du programme, les paramètres, le format des réponses et l'adresse du service.
- *UDDI* (*Universal Description Discovery and Integration* [OASIS-04]) – Un registre *UDDI* représente un annuaire qui contient des informations détaillées concernant les services Web. *UDDI* définit un protocole d'interrogation et de mise à jour d'un tel annuaire. En quelque sorte, *UDDI* propose d'automatiser toute la procédure de recherche et de découverte des services Web.

Nous montrons les relations entre ces trois technologies dans la Figure II.2. L'application cliente désirant interagir avec un service Web doit tout d'abord le localiser. Cette application doit donc effectuer une recherche auprès d'un annuaire en envoyant une requête *UDDI*. Un fournisseur doit ainsi publier son service Web auprès d'un annuaire

UDDI pour qu'il puisse être retrouvé. Une fois le service localisé, l'application cliente peut avoir accès au document *WSDL* décrivant comment contacter le service Web respectif. Le client est ainsi capable de construire un message *SOAP* conforme au format spécifié dans le document *WSDL*. Finalement l'application cliente peut envoyer la requête au service et attendre sa réponse sous le format d'un message *SOAP*.

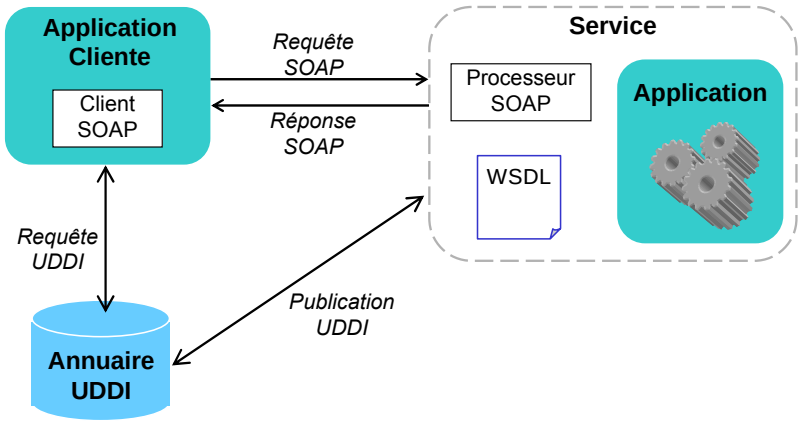

Figure II.2 Exemple d'interaction simple entre une application cliente et un service Web

Remarquons que *SOAP* représente une sorte de façade qui masque l'hétérogénéité technique de l'application ou des données sous-jacentes[1]. Nous avons donc une surcouche logicielle intercalée entre le programme développé (dans un langage propriétaire) et le monde extérieur. L'interopérabilité est ainsi un aspect intrinsèque des services Web. Grâce à leur formalisation standard définissant des normes ouvertes, ils fournissent un moyen technique pour exposer les interfaces de programmation (API, de l'anglais *Application Programming Interface*) de leurs applications dans un format compréhensible par n'importe quelle autre application. Des applications implémentées dans divers langages de programmation et sur diverses plates-formes peuvent ainsi employer des services Web pour interagir et échanger des données à travers des réseaux informatiques.

II.4.2.4. Bilan des solutions générales d'intégration

CORBA, *DCOM* et *Java RMI* ont connu une popularité énorme dans les années 90. Ils étaient considérés comme des solutions de référence pour simplifier le développement d'applications reparties et pour viabiliser l'interopérabilité dans des systèmes distribués et hétérogènes.

L'avantage de *CORBA*, par rapport à ses concurrents, est qu'il a été développé dans un contexte normalisé, puisqu'il a été contrôlé par l'*OMG*, organisme neutre de standardisation. Or, *CORBA* prévoit une mise en œuvre dans différents langages. *RMI* présente le même principe de base que CORBA pour ce qui concerne les appels de méthodes à distance. La différence essentielle est que *RMI* ne s'applique qu'à *Java*. De plus, *CORBA* présente une architecture particulièrement plus riche que *RMI* grâce aux *ORB*s. En conséquence, *CORBA* finit par être beaucoup plus compliqué à mettre en œuvre, c'est la raison pour laquelle de nombreux développeurs se tournent en général vers *RMI*. Quant à *DCOM*, il a été également adopté par des développeurs mais il a toujours souffert

[1] Le format *XML* (textuel) facilite l'échange de données entre systèmes informatiques distincts, chaque information, encadrée par une balise, étant interprétable aussi bien par un programme que par un individu.

des fortes critiques concernant le fait d'être contrôlé par un seul fournisseur et de présenter une dépendance aux plates-formes de *Microsoft*.

Ces intergiciels ont répondu à l'interopérabilité dans le contexte des systèmes distribués de manière assez satisfaisante. Cependant, l'intégration par le biais de ces intergiciels, impose un couplage très fort des applications, ce qui par exemple dans le contexte des entreprises n'est pas vraiment souhaitable vu le dynamisme et l'évolutivité des systèmes d'information. Les solutions d'EAI ont ainsi pris la voie des MOMs et se sont appuyées sur l'échange de messages pour favoriser le découplage entre applications. Mais la mise en œuvre de solutions d'EAI soulève des difficultés associées à l'ampleur du travail d'intégration et au coût des licences logicielles [Alonso-04a]. Quant aux solutions *open sources*, elles ne représentent en général pas des plates-formes complètes et prêtes à l'emploi. Par conséquent, peu de petites et moyennes entreprises se sont lancées dans ce type de projet. L'intégration des applications à l'intérieur d'une entreprise et l'automatisation des interactions avec des partenaires ont été souvent sous-exploitées, voire inexistantes.

Avec l'apparition de l'Internet, une infrastructure complètement ouverte, de nouvelles exigences concernant l'interopérabilité des systèmes sont apparues. Les services Web ont été conçus en premier lieu pour répondre à ces problèmes d'interopérabilité. Actuellement, ils semblent constituer la solution d'avenir pour intégrer des systèmes distribués sur l'Internet. En définissant une architecture orientée service et en s'appuyant sur des normes et protocoles ouverts, ils peuvent réduire considérablement le coût d'intégration d'applications hétérogènes [Stal-02]. Par ailleurs, des solutions d'EAI adoptent à l'heure actuelle des technologies de services Web pour, par exemple, faciliter leur intégration avec d'autres solutions d'EAI.

II.4.3. Plates-formes et environnements pour l'intégration d'applications collaboratives

Les plates-formes et environnements généraux d'intégration décrits dans la section précédente représentent en fait des réponses techniques au problème d'interopérabilité entre différentes applications. Ils offrent des moyens permettant aux applications de communiquer et d'échanger des informations. Néanmoins ces solutions permettent uniquement de maîtriser ce qui est appelé l' "interopérabilité syntaxique" [Lewis-04]. Selon Brownsword *et. al* [Brownsword-04] l'interopérabilité va au delà de la capacité de partager et échanger des informations spécifiques. Les applications doivent se mettre d'accord par rapport à la sémantique de leurs échanges et la façon dont elles doivent réagir.

Dans le domaine du TCAO, les chercheurs se sont rendus compte que cette interopérabilité syntaxique n'était pas suffisante pour construire un environnement collaboratif intégré. Nous trouvons donc dans la littérature certains travaux qui cherchent à faire interopérer différentes applications collaboratives. En se focalisant sur l'intégration de leurs fonctionnalités de collaboration, ces environnements peuvent définir une sémantique derrière cette intégration.

Une caractéristique importante de ces solutions réside dans la réduction d'une partie de la complexité normalement liée au développement d'environnements collaboratifs intégrés. En effet, dans la construction d'un environnements collaboratif intégré à partir d'applications existantes, les fonctionnalités spécifiques à la collaboration seront prises en charge par les applications elles mêmes. Ces environnements ne doivent donc se focaliser que sur le niveau et la sémantique d'intégration à mettre en œuvre.

Toutefois, puisque ces applications n'ont pas forcément été conçues pour être intégrées à un environnement spécifique (par exemple, elles ne s'appuient pas sur un intergiciel), l'intégration peut s'avérer être une tache très complexe.

Des travaux comme celui présenté par Dewan et Sharma [Dewan-99] s'intéressent aux problèmes de base relatifs à l'interopérabilité d'applications collaboratives hétérogènes. Le but n'est pas vraiment d'offrir un environnement intégré où des applications offrant des fonctionnalités complémentaires pourraient être combinées. L'idée concerne plutôt la possibilité de faire interagir des applications similaires mais hétérogènes. En particulier, dans [Dewan-99], les auteurs s'intéressent à l'interopérabilité d'applications collaboratives qui manipulent de manière concurrente les mêmes artefacts (comme des documents textuels). Le travail se focalise sur le couplage des mécanismes de contrôle d'accès (par exemple le *floor control*) employés par des applications distinctes lorsqu'elles accèdent à un même artefact. Ce travail ne s'intéresse par contre pas à l'interopérabilité d'applications qui, bien qu'impliquées dans les mêmes tâches collaboratives, ne manipulent pas les mêmes artefacts (par exemple, une vidéoconférence et un tableau blanc partagé). De plus, l'approche adoptée par ce travail part du principe que à la fois la structure interne et le code source de l'application sont connus.

CVW [Spellman-97] est un prototype d'environnement collaboratif définissant des espaces de travail partagés qui s'appuie sur la métaphore d'un bâtiment réparti en étages et pièces, où chaque pièce définit un contexte différent pour la communication et le partage de documents. Les utilisateurs entrent dans une pièce afin de pouvoir collaborer en utilisant les applications qui ont été intégrées à l'environnement. *CVW*, d'un point de vue technique, se présente comme une sorte de *framework* d'intégration de fonctionnalités collaboratives. L'intégration d'une application peut être accomplie via (i) son interface de programmation (API), (ii) une ligne de commande (permettant simplement de lancer l'exécution de l'application), ou (iii) la modification de son code source. L'intégration n'est cependant pas évidente. La possibilité la plus simple est bien évidement une simple commande d'exécution de l'application, comme c'est le cas de deux applications collaboratives qui ont déjà été intégrées (en l'occurrence *vi*c [vic] et *vat* [vat]). Pour un niveau d'intégration plus fin, où l'application pourrait échanger des donnés avec *CVW* (par exemple, pour qu'un éditeur partagé puisse accéder au serveur de documents de *CVW*), il faut que son *API* soit définie selon des contraintes imposées par l'environnement. L'application doit ainsi être développée selon le cadre spécifié par *CVW*.

Un autre exemple d'environnement collaboratif qui prévoit l'intégration d'applications collaboratives est *DARE* (Activités Distribuées dans un Environnement Réflexif [Bourguin-00]). *DARE* adopte une approche mêlant une architecture cadre (*CORBA* et son modèle de composants) et une implémentation ouverte pour générer des environnements de TCAO. Mais cet environnement est également capable d'intégrer des collecticiels existants de manière isolée, ceux-ci n'ayant pas forcément été développés en vue de cette intégration. Néanmoins, il existe une contrainte imposée aux applications collaboratives pouvant être intégrées dans *DARE* : elles doivent toutes posséder une partie cliente implémentée sous forme d'une *applet Java*. Une des raisons justifiant cette contrainte concerne le choix des auteurs à offrir un environnement où les utilisateurs puissent être mobiles (utilisation d'un simple navigateur Web).

Dans [Iqbal03] les auteurs décrivent trois différents niveaux qui peuvent être atteints lors de l'intégration d'applications collaboratives. Le premier niveau, appelé "intégration de surface" (de l'anglais *surface integration*) correspond à l'intégration de systèmes autonomes qui interopèrent simplement par le biais d'échange de données, soit

directement soit à travers un tableau blanc ou un tableau d'affichage partagés. Le deuxième niveau implique un niveau d'intégration plus élevé où les activités collaboratives supportées par chaque application et les artefacts partagés qui sont considérés comme étant équivalents doivent être fusionnés. Le troisième niveau représente une intégration complète des applications conduisant à un système unique (y compris des interfaces utilisateurs). Tous les conflits, conséquences de la fusion des activités et des objets (ou artefacts) partagés, doivent être résolus.

Selon Iqbal *et al.* les deux dernières niveaux sont les plus envisageables puisqu'ils impliquent une vraie intégration sémantique des applications collaboratives. Afin de rendre possible cette intégration, les auteurs proposent un cadre général pour les systèmes TCAO s'appuyant sur les trois modèles présentés par Ellis et Wainer [Ellis-94] :

- le modèle ontologique spécifie tous les objets partagés dans l'application, ses relations et terminologies ;
- le modèle de coordination spécifie comment les interactions ont lieu dans le système sous la forme d'une sorte de flux de processus ;
- le modèle d'interface utilisateur décrit la manière dont le système est présenté à l'utilisateur.

Le cadre général d'intégration proposé par [Iqbal-03] prévoit ainsi une intégration au niveau de chacun de ces modèles. Le processus d'intégration consiste à identifier, pour chaque application, les éléments associés aux trois modèles et ensuite "fusionner" ces éléments en générant trois modèles intégrés. Encore une fois, une connaissance interne de l'application collaborative s'avère nécessaire pour mettre en correspondance les fonctionnalités offertes par les applications collaboratives avec les trois modèles et pouvoir réaliser l'intégration. En conséquence, l'intégration d'applications préexistantes développées par des tiers devient une tâche très complexe (voire impossible) à réaliser.

Afin d'éviter de devoir tenir compte des détails internes d'implémentation des applications intégrées (facilitant ainsi l'intégration d'applications existantes), certains environnements ont proposé une approche d'intégration faiblement couplée, offrant une d'intégration de surface [Iqbal-03]. Rappelons que ce même type d'approche a été adopté par les solutions générales d'intégration abordées dans la section précédente (par exemple MOMs et services Web). Les motivations sont identiques :

(i) il n'est pas nécessaire d'avoir des connaissances sur les détails des procédures exécutées ou des artefacts traités par les applications collaboratives ;

(ii) une fois intégrées à l'environnement, les applications collaboratives peuvent préserver leur autonomie ;

(iii) des applications peuvent être intégrées et retirées de l'environnement sans que le comportement de base du système soit altéré.

Cette dernière caractéristique est particulièrement intéressante si nous considérons l'importance de la flexibilité pour les systèmes collaboratifs. L'idée est que l'ensemble des applications intégrées à l'environnement puisse être plus aisément modifié.

AREA [Fuchs-99] et *NESSIE* [Prinz-99] ont proposé une intégration faiblement couplée pour mettre en place une notion de présence inter-applications (*cross-application awareness*). L'objectif est de constituer un environnement collaboratif où différentes applications indépendantes puissent partager un espace d'information commun mis en œuvre à travers une infrastructure de notification d'événements. Ainsi, les utilisateurs peuvent recevoir des notifications d'événements provenant de différentes applications (utilisées par d'autres membres du groupe) et pertinents vis-à-vis de l'activité collaborative

globale. *AREA* propose un modèle d'application identifiant les acteurs, les types d'événements, les artéfacts et leurs relations. En s'appuyant sur ce modèle, il est possible de créer, pour chaque utilisateur, une liste de "spécifications d'intérêts" : les événements sont notifiés aux utilisateurs selon cette liste. *NESSIE* utilise un schéma similaire, où on doit spécifier des "profils d'intérêts" pour que le client puisse être notifié de certains types d'événements. Un aspect intéressant de ces deux systèmes est qu'ils utilisent des technologies Internet ouvertes telles que *HTTP* et *CGI* pour permettre l'intégration d'applications existantes. Cependant, la notification d'événements ne sert qu'à mettre en oeuvre une notion de présence et d'activité de groupe aux utilisateurs. En fait, aucun mécanisme n'est prévu pour définir comment une application devrait réagir à des événements notifiés par d'autres applications

Comme dans le cas des solutions générales d'intégration, quelques travaux ont choisi de s'appuyer sur les technologies de services Web pour mettre en œuvre une approche faiblement couplée d'intégration.

XGSP [Fox-03] est un *framework* qui propose l'intégration d'applications de vidéoconférence et audioconférence basées sur les standards *SIP* [Rosenberg-02], *H.323* [*H.323*] et *Access Grid* [AG][1]. Au coeur du *framework* nous avons les serveurs de gestion de collaboration *XGSP* (*XGSP collaboration manager servers*) en charge de gérer une session collaborative en même temps qu'ils contrôlent la création des sessions, la connexion et déconnexion des utilisateurs dans les applications. Pour permettre aux applications intégrées de communiquer avec les serveurs *XGSP*, un protocole commun de signalisation est utilisé. Pour chaque type d'application (*i.e.* les applications utilisant *SIP*, *H.323* et les applications du *Access Grid*), on définit une passerelle capable de faire la traduction entre le protocole de signalisation respectif et le protocole de signalisation *XGSP*. Les services Web sont utilisés pour mettre en œuvre la communication entre les serveurs *XGSP* et les passerelles. La limitation la plus importante de *XGSP* est qu'il prévoit à l'origine uniquement l'intégration d'applications s'appuyant sur *SIP*, *H.323* ou les applications utilisées dans le *Access Grid*. Pour intégrer des applications utilisant d'autres normes de signalisation il est nécessaire de développer les passerelles correspondantes.

Les auteurs de [Dustdar-04] discutent de l'importance des services Web pour atteindre l'interopérabilité entre différents collecticiels. Ils proposent dans cet article une architecture de gestion et de configuration de services Web présentant une couche de services de *teamwork* (*teamwork services layer*). L'objectif de cette couche est d'offrir des interfaces de services, accessibles en tant que services Web, communs aux applications collaboratives (par exemple gestion de groupes, des fichiers et calendriers partagés, *etc.*). Ces services peuvent ainsi être utilisés de façon uniforme par différentes applications intégrées. Pour intégrer des applications collaboratives en s'appuyant sur l'architecture proposée, les auteurs partent du principe que ces applications offrent un support aux technologies de services Web (par exemple *Groove Virtual Office* [Groove]). Cela impose des limitations à l'intégration parce que, bien que l'accès aux applications par le biais d'interfaces de services Web soit rendu de plus en plus disponible, seul un nombre limité d'applications collaboratives le font.

[1] *SIP* (*Session Initiation Protocol*) et *H.323* sont des normes définissant des protocoles de signalisation simples pour les applications de vidéoconférence et audioconférence sur Internet. *Acces Grid* constitue un ensemble d'applications pour mettre en œuvre des sessions de collaboration et des conférences à distance.

II.4.4. Positionnement de notre proposition

Dans la section précédente nous avons pu constater la difficulté associée à la définition d'un environnement pour l'intégration d'applications collaboratives existantes. Les quelques travaux, pas très nombreux, qui ont proposé des solutions pour supporter l'intégration d'applications développées par des tiers, ont été amenés à imposer des contraintes concertant le type de ces applications.

Si d'un coté, il n'est pas concevable de vouloir intégrer toutes les applications, nous sommes convaincus qu'une approche d'intégration faiblement couplée (basé sur l'échange de messages) pourra ouvrir la voie à des applications candidates. Pour mettre en œuvre cette approche, le meilleur choix semble de s'appuyer sur les technologies de services Web. Bien évidement en faisant ce choix, nous devons prendre en compte des applications n'offrant pas de support à ces technologies.

Une solution envisageable, qui commence à être adoptée dans d'autres domaines concernés par l'intégration d'applications, consiste à empaqueter (*wrap*) les applications en utilisant des services Web (par exemple [BEA-04], [Pierce-04]). Néanmoins, un des inconvénients de cette approche est que, selon l'architecture des applications collaboratives existantes, le *wrapping* complet de ces applications en tant que services Web peut conduire à des efforts de développement importants. Un autre problème, particulièrement lié à l'utilisation du protocole *SOAP*, est que cette approche conduit à la définition d'une couche supplémentaire qui peut ralentir le traitement des messages échangés [Chiu-02]. Cela s'explique par le fait que pendant l'échange de messages, la couche de services Web doit empaqueter et désempaqueter toutes les donnés échangées dans le format texte spécifié par *SOAP*.

Dans [Alonso-04b] les auteurs proposent d'utiliser les services Web plutôt pour exécuter des opérations considérées comme étant *coarse-grained*, où l'impact de la surcharge due au traitement des messages *SOAP* est moins important. Conformément à ces recommandations, nous avons décidé de restreindre l'utilisation des services Web dans notre architecture LEICA. Comme nous le verrons dans les chapitres suivants, LEICA propose une architecture hybride où les services Web sont utilisés pendant les étapes initiales d'intégration d'une application, et pendant la création et la mise en exécution de sessions collaboratives intégrées. Ensuite, une deuxième infrastructure est utilisée pour interconnecter les applications durant l'exécution d'une session collaborative intégrée.

Malgré l'aspect faiblement couplé de son infrastructure d'intégration, LEICA va au delà d'une intégration de surface. En fait, l'environnement définit un mécanisme permettant d'associer à une session collaborative intégrée la sémantique désirée de cette intégration. La sémantique est définie par des politiques de collaboration spécifiant le comportement des applications intégrées. Plus précisément, nous spécifions la manière dont une application doit réagir aux interactions des autres applications.

Dans ce mémoire nous allons développer différents aspects de LEICA : l'approche générale d'intégration, la définition de l'architecture et l'implémentation du prototype. Mais avant d'aborder ces aspects, nous allons décrire, dans le prochain chapitre, le point de départ qui a servi de base à la réalisation de nos travaux.

Chapitre III

L'Approche d'intégration initiale

III.1. Introduction

Les travaux développés dans le cadre de cette thèse ont commencé lors de notre participation au Projet Européen *Lab@Future*[1] [Lab@Future]. Ce projet se situait dans le domaine du *E-Learning* et, tout en s'appuyant sur un cadre pédagogique théorique, il a eu pour objectif de développer de nouveaux moyens technologiques permettant, dans le cadre d'un processus d'apprentissage, d'accéder à des expériences de laboratoire réalisées à distance dans différents pays européens.

Le cadre pédagogique s'est appuyé sur le constructivisme social et la théorie de l'activité [Engeström-99] [Mwanza-01], des théories qui défendent que le savoir ne peut pas être directement transféré d'un enseignant vers un apprenant. En fait, l'apprentissage de connaissances représente un processus actif où les apprenants construisent graduellement leurs propres connaissances en testant des idées et des approches préalablement acquises. Autrement dit, l'apprentissage doit être consolidé par des activités pratiques conduisant à la résolution de problèmes et par un raisonnement critique. En même temps, l'apprentissage est également influencé par des interactions résultant de la collaboration entre étudiants, sans ou avec l'intervention de professeurs.

Ainsi, *Lab@Future* était concerné par la possibilité d'appliquer ces théories dans le cadre du *E-Learning*. L'un de ses principaux objectifs était donc de permettre à des groupes d'étudiants d'avoir un accès distant, voire mobile, à des expériences réelles ou virtuelles. Le point de départ du projet était un ensemble d'applications en réalité virtuelle (certaines étant collaboratives) permettant la réalisation d'expériences dans différentes disciplines comme les mathématiques et l'environnement. La première tâche consistait à rendre toutes ces applications collaboratives. L'idée était alors d'exécuter les expériences dans des CVEs en utilisant la technologie apportée par un des partenaires du projet (en l'occurrence *Parallel Graphics* [PG]). Le deuxième défi consistait à augmenter les possibilités de collaboration en intégrant aux CVEs une plate-forme collaborative générale. Le propos de cette plate-forme était d'offrir toutes les facilités de collaboration et de communication pour satisfaire les besoins exprimés par les théories pédagogiques.

Cette problématique visant à étendre un CVE conventionnel (c'est-à-dire, un CVE qui n'assure des fonctionnalités de collaboration entre utilisateurs que par le biais de la réalité virtuelle) en lui intégrant une plate-forme de collaboration nous a incité à faire des recherches dans cette direction. Si dans le cadre du projet une solution *ad hoc* a été mise en

[1] Soutenu par le programme IST de la Communauté Economique Européenne (CEE).

place[1], nous avons, de notre côté, développé une solution plus générale visant à étendre des CVEs existants.

Dans ce chapitre nous présentons l'environnement d'intégration appelé *Collaborative Integration Evironment* (CIE) que nous avons initialement développé dans le but de proposer une solution générale d'extension de CVEs existants par l'intégration d'autres outils de collaboration. Nous commençons par une introduction de cet environnement où nous présentons nos motivations et l'approche d'intégration adoptée lors de sa spécification. Ensuite nous donnons des détails sur le prototype développé pour finalement présenter quelques conclusions obtenues à la lumière de cette première expérience d'intégration.

III.2. CIE : l'environnement d'intégration collaboratif

L'idée derrière le CIE est de permettre que différents outils de collaboration soient intégrés à un CVE conventionnel. L'objectif principal de cette intégration est de permettre que la collaboration ne se restreigne pas au contexte de la réalité virtuelle. L'intuition de base consiste à combiner différentes fonctionnalités offertes par des outils de collaboration (édition de documents, communication entre personnes, *etc.*) au sein d'un monde virtuel partagé.

III.2.1. Les contraintes des CVEs conventionnels

Les CVEs constituent depuis quelques années un sujet de recherche en plein essor, et plusieurs technologies pour le développement d'environnements virtuels collaboratifs ont été décrites dans la littérature.

Des systèmes comme *COSMOS-2* [Darlagiannis-00], *MASSIVE-3* [Greenhalgh-00] et *NPSNET-V* [Capps-00] proposent des cadres généraux et/ou des librairies applicatives pour faciliter le développement de mondes virtuels partagés en trois dimensions (3D). Ces différentes solutions traitent les problèmes importants spécifiques aux CVEs (propagation d'état, coordination, cohérence, *etc.*). Ces trois systèmes offrent des moyens pour étendre le CVE uniquement par le chargement dynamique de nouvelles entités 3D. Mais l'intégration d'autres fonctionnalités de collaboration/communication n'a pas été prévue (dans le cas de *MASSIVE-3*, des communications audio entre utilisateurs peuvent être supportées).

Même si certains des systèmes décrits précédemment définissent leurs propres mécanismes d'extension, ils permettent uniquement d'intégrer des objets ou composantes 3D directement à l'intérieur du monde virtuel. Parmi les CVEs existants, seules quelques solutions proposent d'intégrer l'activité collaborative en cours au sein du monde virtuel avec d'autres applications. En particulier, quelques systèmes offrent des mécanismes d'interface avec le Web. En effet, étant donné que le Web devient aujourd'hui la plate-forme de communication et de collaboration la plus importante, il est crucial d'aborder ce problème.

Dans [Pekkola-00], par exemple, les auteurs proposent l'intégration de VIVA, qui est un CVE, à *CRACK!*, qui est un outil permettant à l'origine de rendre compte de la présence des autres utilisateurs visitant la même page WEB. L'idée consiste à associer à

[1] La plate-forme Platine [Platine], constituée de différents outils de communication et de collaboration, a été directement intégrée aux CVEs correspondant aux expériences.

des pages Web des mondes virtuels. Un utilisateur visitant ces pages est ainsi au courant des autres utilisateurs visitant les mêmes pages ou de ceux qui sont connectés aux mondes virtuels associés. Celui qui est connecté à un monde virtuel peut également visualiser ceux qui naviguent sur le Web. Au delà de l'intégration avec le Web, *VIVA* intègre d'autres media permettant aux utilisateurs de communiquer entre eux dans d'autres contextes que celui de la réalité virtuelle, tels que l'audio, la vidéo et le chat.

Un autre système qui supporte la collaboration au delà d'un monde virtuel est le CVE présenté dans [Frécon-98]. Ce CVE s'interface avec un éditeur de texte qui est associé à la représentation 3D d'un bloc-notes. Les utilisateurs peuvent sélectionner le bloc-notes pour lire ou éditer des textes. En plus de l'éditeur de texte, le monde virtuel dispose d'une métaphore d'un système de courrier électronique permettant aux utilisateurs de poster des documents encapsulés dans des enveloppes 3D à destination d'autres utilisateurs qui ne sont pas connectés à l'environnement.

Blaxxun Platform [Blaxxun] définit un mécanisme de partage de documents intégré à un environnement virtuel. Ce mécanisme permet de présenter à l'intérieur du monde virtuel des documents de différents formats. Un outil de communication VoIP (*Voice over IP*) est également inclus pour permettre à des utilisateurs connectés de communiquer entre eux.

Remarquons que, malgré le fait d'étendre des mondes virtuels en offrant d'autres moyens de collaboration (en dehors de celui de la réalité virtuelle), les systèmes présentés dans [Pekkola-00], [Frécon-98] et [Blaxxun] sont des systèmes figés et donc pas extensibles. De manière *ad hoc*, ces systèmes ont intégré des fonctionnalités de collaboration supplémentaires aux CVEs sans chercher à offrir des solutions générales d'intégration.

L'environnement *MOVE* [García-02] semble constituer une solution plus générale pour intégrer des applications externes à un CVE. Il définit une infrastructure extensible (s'appuyant sur l'architecture *ANTS*) permettant à des développeurs de créer et d'insérer de nouveaux outils dans le système. Cependant, la limitation principale de cette approche est qu'elle ne permet que l'intégration d'applications conçues spécifiquement dans ce but.

Comme nous l'avons souligné, le problème de l'intégration d'applications externes à des CVEs n'a pas jusqu'à maintenant été traité de manière satisfaisante, même si potentiellement cette intégration est très fortement souhaitable. Ainsi le développement de notre travail a été motivé au départ par la constatation du fossé existant entre la plupart des CVEs actuels et d'autres applications collaboratives. Partant de ce constat, nous avons eu comme but de définir un environnement qui soit le plus général possible pour l'intégration d'outils de collaboration externes à des CVEs, permettant ainsi d'enrichir des CVEs conventionnels.

III.2.2. Le cadre général d'intégration

Pour réaliser l'intégration d'applications collaboratives qui sont à l'origine indépendantes, nous avons défini le CIE qui s'appuie sur un cadre général d'intégration, dans lequel différentes applications client/serveur peuvent s'intégrer comme cela est illustré dans la Figure III.1. Nos deux besoins de base pour réaliser cette intégration sont : (i) l'indépendance vis-à-vis de la plate-forme et (ii) l'extensibilité. Le premier besoin découle de l'hétérogénéité des systèmes répartis actuels. Le deuxième besoin vise à assurer que l'environnement ne se restreindra pas à des scénarios d'intégration prévus à l'avance. La décision de ne considérer que des systèmes client/serveur a été motivée par le fait que les CVEs fonctionnaient, à l'époque, la plupart en mode client/serveur.

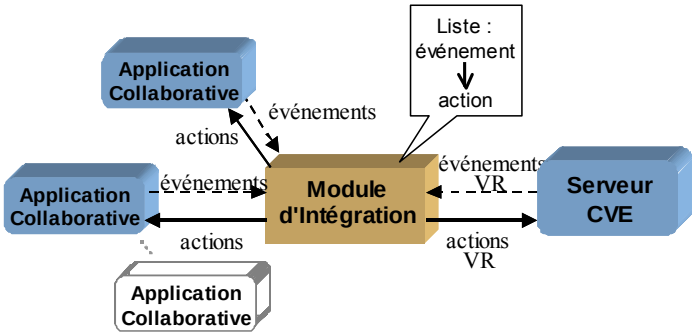

Figure III.1 Cadre général d'intégration

Dans cet environnement, les applications collaboratives doivent être interfacées par un *Module d'Intégration*. Sa tâche consiste fondamentalement à récupérer les événements notifiés par chaque serveur d'application (ainsi que les interactions utilisateur) et à exécuter des actions par l'intermédiaire de ces serveurs. Ce module permet d'associer des événements produits dans un contexte de collaboration à des actions qui seront exécutées dans un autre contexte de collaboration. Cette association est réalisée au moyen d'une liste statique mettant en relation événements et actions.

III.2.3. L'architecture de base

En nous appuyant sur le cadre général d'intégration, nous avons défini une architecture répartie dont le cœur est l' *IntegrationModule*[1]. À travers une interface de communication bien définie, ce module peut contrôler toute l'activité de collaboration ayant lieu dans les différents contextes de collaboration.

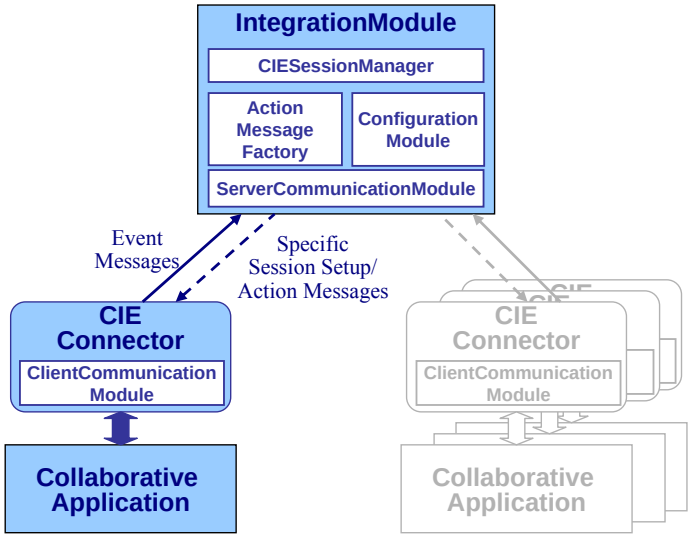

Figure III.2 L'architecture générale du CIE

La Figure III.2 illustre l'architecture de base de l'environnement d'intégration. Du côté du serveur d'application, nous avons le *CIEConnector* dont l'élément principal est le *ClientCommunicationModule* qui est utilisé par ce serveur pour (i) s'enregistrer auprès du *IntegrationModule* et pour (ii) lui envoyer des *EventMessages* qui permettent à celui-ci de suivre l'activité de collaboration gérée par ce serveur. Du côté de l'*IntegrationModule*

[1] Les modules de l'architecture sont nommés en anglais.

existe un sous-module, le *ServerCommunicationModule*, qui attend l'enregistrement de chaque serveur d'application. Lorsqu'un serveur se connecte, l'*IntegrationModule* l'enregistre et reste ensuite en attente des *EventMessages*. Chaque fois qu'un *EventMessage* arrive, le sous-module *CIESessionManager* vérifie s'il existe des actions associées (pour chaque session collaborative une liste[1] a été définie où chaque élément associe un *EventMessage* à un ensemble de *ActionMessages*). Si c'est le cas, il appelle le *ActionMessageFactory* pour construire les *ActionMessages* correspondants (s'appuyant sur certains paramètres). Cet *ActionMessage* est envoyé au serveur d'application cible, qui le reçoit par l'intermédiaire de son *ClientCommunicationModule* et le traite ensuite.

III.3. Implémentation de l'environnement d'intégration collaboratif

Pour implémenter le prototype du CIE nous avons choisi deux applications collaboratives à intégrer par le module d'intégration : (i) *Vnet*, représentant un CVE, et (ii) *CoLab*, une application collaborative développée au LAAS-CNRS. Remarquons que les serveurs des applications doivent être *open source* afin de pouvoir intégrer le *CIEConnector*.

Vnet[2] est une application client/serveur implémentée en Java assurant le partage de scènes VRML, et permettant ainsi la création de mondes virtuels collaboratifs. La communication entre chaque client et le serveur est réalisée par le protocole VIP (*VRML Interchange Protocol*) qui permet d'échanger des champs VRML[3] [VRML]. Du côté client, existe une *applet* Java qui communique par EAI (*External Authoring Interface*) avec un *plug-in* VRML. Cette *applet* est responsable de détecter des changements d'état dans les objets VRML et de les communiquer au serveur. Une fois que ces changements ont été notifiés, le serveur les diffuse aux autres *applet*s qui mettent localement à jour leur monde virtuel.

CoLab [Hoyos-02] est un environnement logiciel s'appuyant sur Java qui met en œuvre un système de navigation Web collaborative. A l'époque du développement de ce prototype, *CoLab* se trouvait dans une version *Beta* où : le serveur *CoLab* comprenait essentiellement un Web *Proxy* responsable du suivi de l'activité de navigation (au courant des actions de navigation de chaque utilisateur) ; le client *CoLab* représentait une *applet* Java par laquelle le serveur pouvait synchroniser la présentation de pages Web. De cette manière, *CoLab* permet à un groupe d'utilisateurs de naviguer ensemble sur le Web où un utilisateur *Maître* dirige l'activité de navigation. Le rôle de *Maître* est dynamique, c'est-à-dire, celui qui est le *Maître* peut passer ce rôle à un autre membre du groupe.

III.3.1. Implémentation des modules de l'architecture

Comme Java est un langage orienté objets indépendant à priori de toute plate-forme, nous l'avons choisi comme technologie de base pour l'implémentation du prototype. Ainsi, pour implémenter l'environnement d'intégration collaboratif, nous avons défini trois *packages* Java qui comprennent l'ensemble des classes et interfaces nécessaires au développement du système.

---

[1] Cette liste est définie dans un fichier d'initialisation, chargé par le *ConfigurationModule*.

[2] Le système était disponible à l'adresse http://www.csclub.uwaterloo.ca/u/sfwhite/vnet (visitée la dernière fois en juillet 2004) mais n'est actuellement plus disponible.

[3] *Virtual Reality Modeling Language* : langage normalisé pour la description d'objets et de scènes 3D.

Le premier *package* (Figure III.3), *cie.distributed.core*, contient toutes les classes à être utilisées pour le développement du cœur de l'environnement :

i) *IntegrationModule* et *ConfigurationModule* - Le *IntegrationModule* représente le module d'intégration lui-même. Quand un objet *IntegrationModule* est créé, il demande au *ConfigurationModule* de charger un fichier d'initialisation[1]. Ce fichier définit, pour chaque serveur d'application, quels sont les événements qui doivent déclencher des actions. S'appuyant sur ces informations, l'*IntegrationModule* sait quel type d'*ActionMessage* il doit demander de créer à l'*ActionMessageFactory* une fois qu'un *EventMessage* arrive.

ii) *ServerCommunicationModule* - Cette classe utilise *JSDT 2.0*[2] [JSDT] pour offrir les moyens de communication nécessaires. Elle peut commencer et arrêter la session *JSDT* et, pendant que cette session est active, elle tient le rôle de *listener* de messages. Chaque fois qu'elle reçoit un *EventMessage*, elle le délivre à l'*IntegrationModule*. La méthode *sendActionMessage()* est utilisée par l'*IntegrationModule* pour envoyer tous les *ActionMessage*s créés par l'*ActionMessageFactory*.

iii) *ActionMessageFactory* - Cette classe crée tous les *ActionMessage*s possibles. Quand l'*IntegrationModule* appelle le *createActionMessage*(), il passe le type de l'*ActionMessage* à créer ainsi que deux objets de la classe *ParamList*, chacun contenant une liste de paramètres : le première objet contient une liste extraite du fichier d'initialisation, et le deuxième objet contient une liste de paramètres extraite de l'*EventMessage* qui a déclenché le mécanisme de création. S'appuyant sur ces données, l'objet *ActionMessage* correspondant est créé.

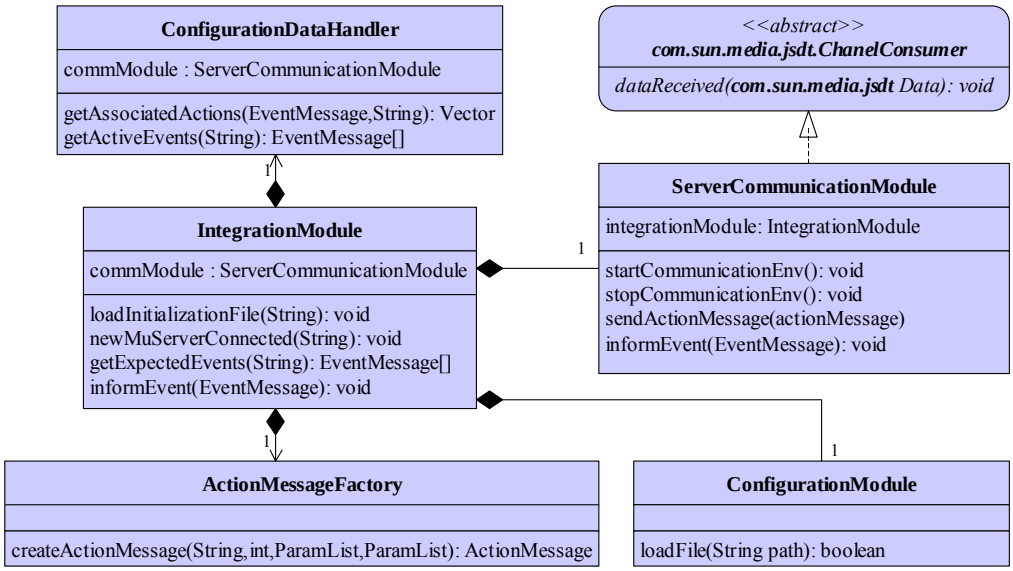

Figure III.3 Le diagramme de classes du package cie.distributed.core

Le deuxième *package* (Figure III.4), *cie.distributed.edge*, inclut les classes et interfaces que chaque serveur d'application doit utiliser pour être intégré dans l'environnement :

i) *MUServerInterface* - Chaque *ServerClass* (correspondant dans le prototype au serveur *Vnet* et au serveur *CoLab*) doit implémenter cette interface Java afin de pouvoir communiquer avec son *ClientCommunicationModule*.

[1] Le prototype ne supporte qu'une session collaborative à la fois.
[2] *Java Shared Data Toolkit* : API permettant d'ajouter des caractéristiques collaboratives aux applications Java.

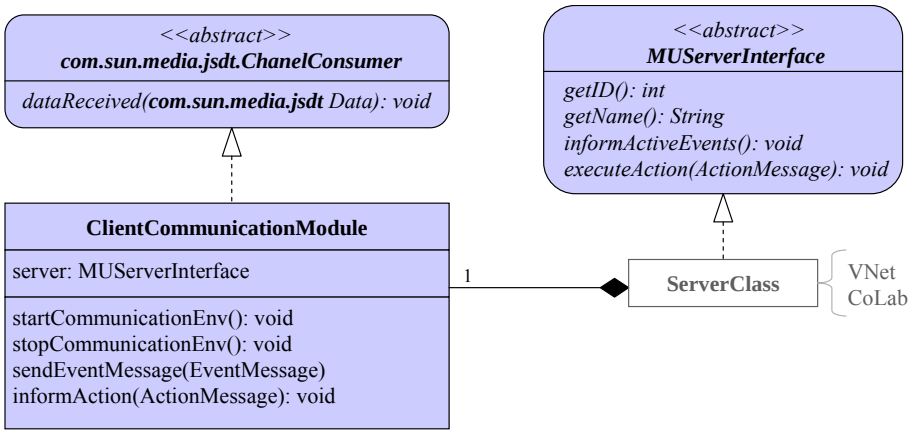

Figure III.4 Le diagramme de classes du package cie.distributed.edge

ii) *ClientCommunicationModule* - cette classe permet au *ServerClass* de communiquer avec le cœur de l'environnement. Pendant que l'activité de collaboration progresse, le *Server Class* crée des objets *EventMessage* et les envoie par l'intermédiaire de ce module. Ce module est également un *listener* des messages. Chaque fois qu'il reçoit un *ActionMessage*, il le délivre au *ServerClass*.

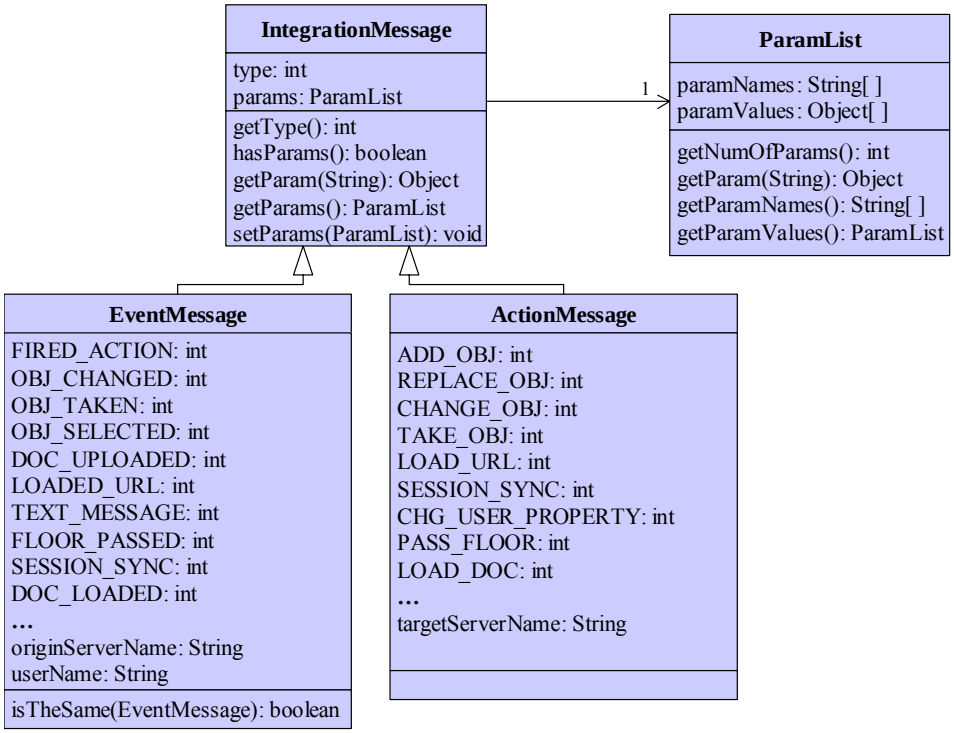

Figure III.5 Le diagramme de classes du package cie.data

Le dernier *package* (Figure III.5), *cie.data*, rassemble les classes utilisées pour modéliser les types de données échangées dans le CIE :

i) *IntegrationMessage* - C'est la super-classe de la classe *EventMessage* et de la classe *ActionMessage*. Elle modélise le comportement commun à ces messages tels que les méthodes d'accès de base et les listes de paramètres qu'ils peuvent véhiculer.

ii) *EventMessage* - Dans cette classe nous définissons tous les types d'événements qui peuvent être produits par tous les serveurs d'application. Une autre spécialisation par

rapport à la super-classe *IntegrationMessage* est relative à l'attribut indiquant le nom de l'utilisateur qui a produit l'événement.

iii) *ActionMessage* - Dans cette classe nous définissons tous les types d'actions qui peuvent être créées par l'*ActionMessageFactory*.

iv) *ParamList* - Une structure de données générique. Chaque *Event*/*ActionMessage* contient un *ParamList* utilisé pour échanger des données.

III.3.2. Le fichier d'initialisation

Comme nous l'avons mentionné précédemment, un fichier d'initialisation est utilisé pour configurer initialement le module d'intégration. Pour permettre la spécification du fichier d'initialisation, nous avons défini un nouveau langage basé sur *XML* appelé *Integration Module Configuration Language* (*IMCL*). Fondamentalement, ce langage autorise la description d'une liste qui associe des événements à des actions.

Dans la Figure III.6, nous illustrons un exemple de fichier (le fichier utilisé dans notre prototype) qui montre la syntaxe de l'*IMCL*. Un fichier *IMCL* contient une liste de groupes "<events>". Chaque "<events>" est associé à un serveur d'application différent grâce au paramètre "from". Dans un groupe "<events>", nous avons la liste des "<event>" représentant tous les événements qui ont comme origine ce serveur d'application.

```
<imcl>
   <events from="CoLab">
      <event type="LOADED_URL">
         <parameters>
            <param name="url"/>
         </parameters>
         <actions>
            <action to="Vnet" type="SET_VALUE">
               <parameters>
                  <param name="obj_name" value="url_string" />
                  <param name="field_name" value="str" />
                  <param name="field_value" value="#url" />
               </parameters>
            </action>
          <actions>
      </event>
   </events>
   <events from="Vnet">
      <event type="OBJ_TAKEN">
         <parameters>
            <param name="obj_name" value="floor_token" />
            <param name="user_name" />
         </parameters>
         <actions>
            <action to="CoLab" type="PASS_FLOOR">
               <parameters>
                  <param name="target" value="#user_name" />
               </parameters>
            </action>
         </actions>
      </event>
   </events>
   ...
</imcl>
```

Figure III.6 Le fichier d'initialisation utilisé dans le prototype

Quand le module d'intégration reçoit un *EventMessage* d'un serveur d'application, il vérifie si le groupe "<events>" associé à ce serveur contient un élément "<event>" du même type (indiqué par son paramètre "type") du message. Si c'est le cas, le module d'intégration vérifie si les paramètres portés par cet objet *EventMessage* ont les mêmes valeurs que les paramètres décrits dans la balise "<parameters>" de ce "<event>". Si les paramètres sont bien identiques, l'*IntegrationModule* crée un *ActionMessage* (en appelant l'*ActionMessageFactory*) pour chaque élément "<action>" listé dans la balise "<actions>" de ce "<event>".

Le scénario d'intégration que nous avons implémenté dans ce prototype a eu pour objectif de représenter dans le monde virtuel de *Vnet* ce qui se passait dans la session de navigation Web collaborative de *CoLab*. Les interactions entre *Vnet* et *CoLab* ont été directement définies par la spécification des associations événement-actions du fichier d'initialisation. Tout utilisateur connecté dans *CoLab* était associé à un avatar[1] dans le monde virtuel. En plus, un objet 3D en forme de sceptre était utilisé pour représenter le *floor* de la navigation Web. Ainsi, l'avatar qui tenait le sceptre dans *Vnet* correspondait à celui qui avait le rôle de *Maître* dans *CoLab*. Par exemple, à travers la dernière association événement-actions qui apparaît dans le fichier illustré dans la Figure III.6, nous avons implémenté le passage du *floor* de la navigation Web dans le monde virtuel. En fait, comme nous l'avons illustré dans la Figure III.7, à chaque fois que le sceptre est passé d'un avatar à l'autre, *Vnet* doit notifier le *EventMessage* correspondant (de type *OBJ_TAKEN*). L'*IntegrationModule* identifie qu'il existe une action (de type *PASS_FLOOR*) associée, et suite à la notification de l'événement, il envoie un *ActionMessage* à *CoLab*. Ce dernier exécute l'action correspondante et force le passage du rôle de *Maître* de la navigation Web à l'utilisateur indiqué.

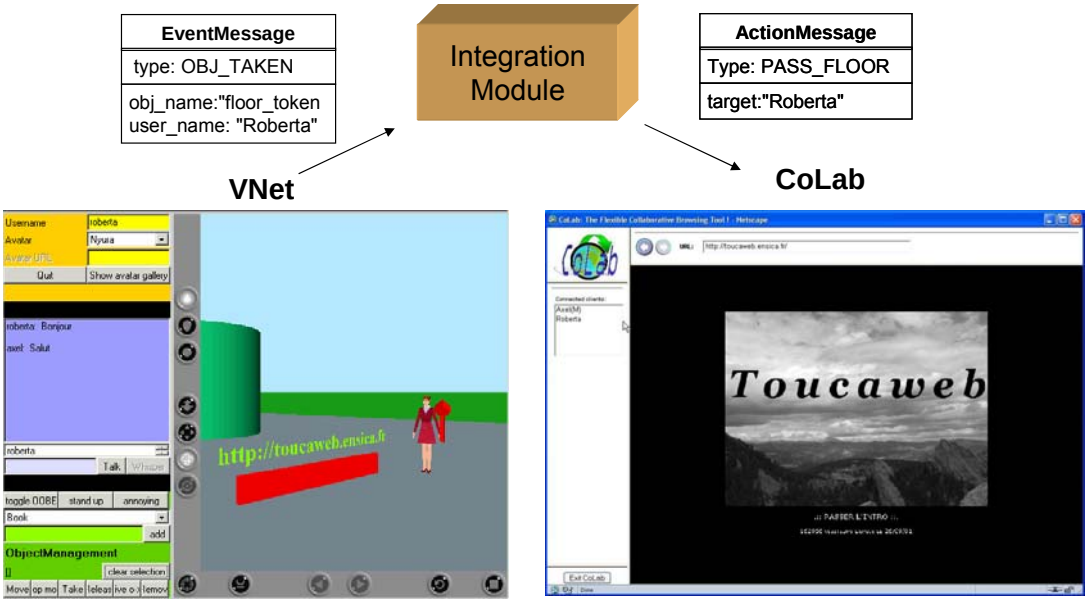

Figure III.7 Le passage du floor de la navigation Web *à travers le monde virtuel*

III.3.3. L'extensibilité de l'environnement

En plus d'être *open source*, deux contraintes de base doivent être satisfaites par les applications collaboratives pour faciliter leur intégration au CIE. Premièrement, il est désirable qu'elles soient implémentées en Java. De cette façon, le système, dans son ensemble, garde son indépendance vis-à-vis de la plate-forme. La deuxième contrainte est due au fait que, puisque nous avons décidé de considérer seulement le paradigme client/serveur, l'architecture de l'environnement a été conçue pour permettre la communication entre les serveurs et le module d'intégration.

Pour intégrer une application qui satisfasse aux contraintes précédentes, nous devons réaliser trois étapes d'intégration :

[1] Personnage utilisé pour représenter un utilisateur dans le monde virtuel.

i) La première étape concerne la définition de tous les types d'événement que cette application peut produire et communiquer au module d'intégration, ainsi que de tous les types d'action que cette application peut recevoir de ce module. Une fois ces types d'événement et d'action définis, ils doivent être inclus dans les classes *EventMessage* et *ActionMessage* respectivement. Remarquons que ceci implique de recompiler les classes que nous avons décrites dans le paragraphe III.3.1.

ii) La deuxième étape doit être exécutée du côté du serveur d'application. Ce serveur (que nous avons appelé *ServerClass* dans la description de l'architecture) doit implémenter la *MUServerInterface* et créer un objet de la classe *ClientCommunicationModule*. En implémentant la méthode *createEventMessage()* de la *MUServerInterface*, le *ServerClass* doit traiter la création des différents types d'objets *EventMessage*s qui ont été définis à l'étape précédente. Dans l'implémentation de la méthode *MUServerInterface.executeAction()*, le *ServerClass* doit traiter l'exécution des actions qui correspondent aux différents types d'*ActionMessages* également définis à l'étape précédente.

iii) La dernière étape doit être exécutée dans le cœur de l'environnement. Dans l'*ActionMessageFactory,* il faut tout simplement ajouter le code nécessaire pour traiter la création des nouveaux types d'*ActionMessage*s.

Il est important de remarquer que ce mécanisme d'extension permet à notre environnement d'intégration collaboratif de ne pas être restreint au CVE que nous avons choisi d'étendre. D'autres CVEs (*open sources*) pourraient être étendus à sa place, ou même coexister avec lui.

III.4. Conclusions

Nous venons de présenter dans ce chapitre le contexte dans lequel nos travaux de thèse ont démarré, lors de notre participation au Projet Européen *Lab@Future*. Ce projet, qui cherchait à développer un environnement collaboratif en intégrant de façon *ad hoc* un CVE conventionnel à une plate-forme de collaboration générique, nous a incité à proposer un environnement d'intégration plus général. Cet environnement, que nous avons appelé CIE, visait ainsi l'extension de CVEs conventionnels par l'intégration d'autres collecticiels existants.

S'agissant de notre premier essai concernant l'intégration d'applications, le CIE représente en fait une solution très limitée :

- le fait de ne pouvoir intégrer que des applications *open sources* restreint considérablement les possibilités d'intégration, vu que la plupart des applications sur le marché ne rendent pas disponibles leurs code source ;
- le fait de ne pas prévoir l'intégration d'applications *peer to peer* (P2P[1]) est également contraignant, car, de plus en plus, les applications collaboratives suivent ce mode de répartition ;
- l'utilisation d'une architecture centralisée, supervisée par un module central d'intégration, présente énormément d'inconvénients, notamment pour ce qui concerne le passage à l'échelle de l'environnement ;

[1] Ce terme, traduit par "pair à pair" ou "égal à égal", désigne un modèle architectural où les nœuds ne jouent pas exclusivement les rôles de client ou de serveur. En fait, ils peuvent en fait fonctionner soit en tant que client ou en tant que serveur.

- le principe d'une extensibilité statique, demandant la recompilation de l'environnement à chaque fois qu'une nouvelle application est intégrée, rend l'environnement très dépendant des applications intégrées, impliquant des limitations en termes de modularité, de fiabilité et de passage à l'échelle ;
- finalement, la définition d'un environnement d'intégration visant uniquement l'extension de CVEs finit par limiter les scénarios d'intégration possibles ; différentes applications pourraient être combinées afin de composer un environnement collaboratif intégré sans forcement imposer la présence d'un CVE parmi ces applications.

Malgré les limitations présentées par le CIE, nous avons énormément appris lors de son développement. Cette expérience nous a vraiment aidé dans la définition d'une nouvelle approche d'intégration, beaucoup plus générale et plus élaborée que celle utilisée pour le CIE. En s'appuyant sur cette approche d'intégration, nous avons développé LEICA, un nouvel environnement pour l'intégration d'applications collaboratives, dont les concepts, l'architecture et l'implémentation sont présentés dans les chapitres suivants.

Chapitre IV

L'environnement d'intégration : LEICA

IV.1. Introduction

LEICA représente un environnement d'intégration dont le rôle principal est de rendre possible l'utilisation de différents outils de communication et de collaboration dans le cadre d'une même activité collaborative. Une application collaborative, conçue indépendamment de LEICA, et une fois intégrée à cet environnement, sera capable d'interagir avec l'environnement, et bien entendu, avec les autres applications également intégrées. Le but de cette intégration est de pouvoir coordonner de manière contrôlée différentes fonctionnalités des applications, en allant ainsi bien au delà d'une simple utilisation simultanée de ces applications.

Pour réaliser une activité collaborative au sein de LEICA, nous devons définir et configurer ce que nous appelons une *SuperSession*. Une *SuperSession* représente une session de travail intégrée (globale) comprenant tous les acteurs d'une activité collaborative (*i.e.* les applications collaboratives, les utilisateurs, leurs rôles, *etc.*). Pour bien identifier ces différents éléments, nous avons défini un modèle de *SuperSession*.

Dans le processus de configuration d'une *SuperSession* nous précisons quelles sont les applications collaboratives qui seront utilisées ainsi que leur comportement. Ce comportement se caractérise par les interactions définies entre les applications, interactions qui sont contrôlées par un ensemble de règles de collaboration (ces règles définissent la politique de collaboration associée à une *SuperSession*).

Dans ce chapitre nous allons tout d'abord présenter l'approche générale d'intégration proposée par LEICA, en illustrant initialement quelques scénarios d'intégration afin de mettre en valeur l'importance d'un tel environnement. Par la suite, nous présenterons la définition d'une *SuperSession* en détaillant les différents éléments du modèle proposé. Nous expliquerons ensuite le processus de configuration des *SuperSessions*, en détaillant la spécification des politiques de collaboration.

IV.2. Approche générale d'intégration

L'approche d'intégration employée par LEICA s'appuie sur une stratégie faiblement couplée : les applications, développées à l'origine en tant qu'outils indépendants, sont intégrées à l'environnement tout en gardant leur autonomie. Mais avant d'expliquer comment LEICA met en place cette approche d'intégration, nous allons illustrer, au moyen de quelques scénarios, le type d'intégration que nous envisageons ainsi que le gain apporté par la mise en place d'un tel environnement intégré.

IV.2.1. Scénarios d'intégration

IV.2.1.1. Outil de navigation coopérative enrichi d'un outil de messagerie instantanée (Chat)

CoLab [Hoyos-05a] [Hoyos-05b] est un outil de collaboration développé au sein du groupe OLC qui permet à des utilisateurs de naviguer en groupe sur le Web. Lors qu'un utilisateur rejoint une session de navigation coopérative, il démarre son activité de navigation de façon indépendante (ou asynchrone). Au cours de la session, des relations de synchronisation entre utilisateurs peuvent être créées : un utilisateur peut choisir de suivre (ou de se faire suivre par) un autre utilisateur. Par conséquent, des "arbres de navigation" se forment, chaque arbre représentant un groupe d'utilisateurs (un nœud correspond à un utilisateur) dont l'activité de navigation est guidée par sa racine.

Un scénario d'intégration envisageable est d'intégrer à *CoLab* un outil de messagerie instantanée permettant aux utilisateurs appartenant au même arbre de navigation de communiquer entre eux. Dans ce cas, chaque arbre de navigation est associé à un salon de discussion d'un Chat ; ainsi si , par exemple, un utilisateur quitte un arbre de navigation pour passer à un autre, il est automatiquement transféré, au sein de l'outil Chat, d'un salon de discussion à un autre.

IV.2.1.2. Réunions virtuelles

Les réunions de groupe représentent un exemple très connu d'activité collaborative. Ce genre d'activité s'effectue en général en plusieurs étapes, pouvant être assistées par différents outils de support de réunion et/ou de communication, comme par exemple :

(i) un système d'agenda partagé pour gérer les emplois du temps des membres du groupe, ainsi que le planning de leurs réunions ;
(ii) des outils d'audio et de vidéoconférence ;
(iii) un outil d'édition partagée pour permettre la création et la modification de documents pendant les réunions ;
(iv) et finalement, considérant que les réunions peuvent se réaliser dans le cadre d'un processus métier (appelé également processus opérationnel), un outil de *workflow* pour gérer les documents créés au cours des réunions.

Ainsi, dans un environnement intégré, le système d'agenda partagé se chargerait, selon l'emploi du temps de chaque personne, de planifier la réunion virtuelle et de démarrer automatiquement les outils de audio/vidéoconférence et d'édition partagée en début de réunion. L'outil d'édition partagée pourrait se charger également de passer à l'outil de *workflow* chaque document engendré.

IV.2.1.3. E-learning

Le *E-learning* est le nom donné à l'utilisation des technologies de l'information et de la communication dans la formation initiale et/ou professionnelle. En bénéficiant des avantages des nouvelles technologies, on cherche à rendre plus efficaces les processus d'apprentissage et l'accès à la connaissance.

Puisque les technologies de TCAO ont été conçues pour simplifier le partage d'information, favoriser la communication de groupe, améliorer la coordination des activités et motiver l'implication individuelle, le déploiement de ces technologies dans le

E-learning permet de promouvoir un apprentissage coopératif [1] . Les étudiants communiquent en utilisant des formes d'interaction qui peuvent conduire à la stimulation de mécanismes d'apprentissage, tout en éliminant les inconvénients de la déshumanisation de la formation "à distance".

Ce raisonnement est un des principaux fondements des nouvelles théories pédagogiques telles que le constructivisme et la théorie de l'activité [Engeström-99] [Mwanza-01]. Le constructivisme représente l'approche selon laquelle le savoir est construit par chaque individu grâce à ses interactions avec un environnement social (*i.e.* d'autres individus et les objets). En complément, la théorie de l'activité propose que l'apprentissage des l'élèves se fasse à partir d'une activité conjointe réflexive qui est en constante construction (suite à différents phénomènes comme celui de la coopération). La "situation" n'est jamais donnée à l'avance, mais elle est le résultat d'interactions. Ainsi, l'impossibilité de réellement prédire les besoins nous incite à dépasser les limites d'une approche à priori des environnements collaboratifs et à adopter la conception de systèmes évolutifs permettant l'intégration dynamique de nouveaux outils de collaboration.

Différentes applications collaboratives peuvent faire partie d'un tel environnement. Par exemple, les CVEs (*Collaborative Virtual Environments*) représentent une catégorie importante de systèmes collaboratifs pour le support au *E-learning*. La perception des activités dans les environnements scolaires ainsi que la reproduction des métaphores du monde réel, rendues possibles par la réalité virtuelle, peuvent être considérées comme des exigences pour l'apprentissage en groupe.

Nous pouvons imaginer un scénario où un CVE est utilisé pour mettre en place un monde virtuel représentant le bâtiment d'une école, constitué : d'un hall d'entrée, de deux salles de cours virtuelles et d'une salle des enseignants. Trois types d'utilisateurs peuvent se connecter: "Professeur", "Moniteur" et "Elève". Un outil de messagerie instantanée est intégré au monde virtuel de façon à connecter tous les utilisateurs dont les avatars se trouvent dans le hall d'entrée. Les cours sont présentés à l'aide d'un outil de navigation Web coopérative comme *CoLab* et d'un outil d'audioconférence. Nous associons ainsi à chaque salle de cours une session *CoLab* et une autre session d'audioconférence. Pour suivre un cours, un élève doit quitter le hall d'entrée et entrer dans une des deux salles de cours, ce qui le connecte automatiquement aux outils de collaboration associés. La salle des enseignants sert à réaliser des réunions entre enseignants et moniteurs. Un tableau blanc partagé est employé en tant qu'outil de discussion et dès qu'un utilisateur (jouant un de ces deux rôles) entre dans la salle, il est connecté à la session de tableau blanc.

La Figure IV.1 illustre ce scénario d'intégration avec 7 utilisateurs connectés : à gauche, un plan du monde virtuel, et à droite, un schéma décrivant l'ensemble des applications utilisées pendant la session collaborative intégrée de *E_learning*, chaque application comprenant un ensemble de sessions collaboratives qui lui sont spécifiques.

Notre but n'est pas vraiment d'anticiper tous les scénarios d'intégration possibles. En réalité, cela serait tout aussi impossible que de développer un environnement de collaboration assurant *a priori* toutes les fonctionnalités de collaboration envisageables. Par conséquent, un environnement d'intégration doit être le plus général et le plus flexible possible pour ne pas imposer de contraintes sur l'ensemble des applications pouvant être intégrées.

[1] Le *E-learning* coopératif est à l'origine du domaine de recherche appelé *Computer Supported Cooperative Learning*, ou CSCL.

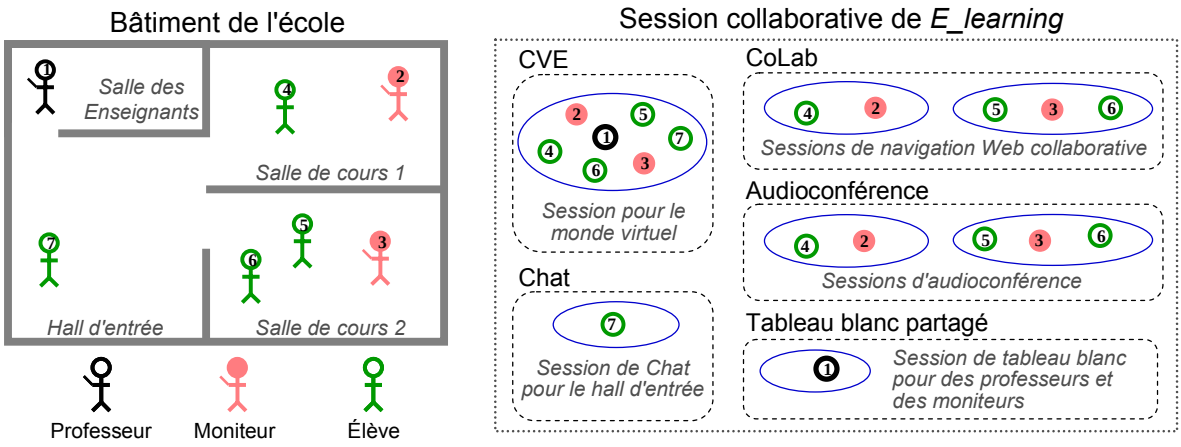

Figure IV.1 Illustration du scénario d'intégration de E_learning

IV.2.2. Cadre général d'intégration

Nous constatons, dans les scénarios décrits précédemment, que les interactions entre les différentes applications se caractérisent par des réactions (ou actions) suite à l'occurrence de certains événements. Ce schéma d'interaction nous a amené à concevoir LEICA en tant qu'environnement d'intégration où les applications peuvent justement s'échanger des informations par l'intermédiaire de notifications d'événement. En fonction de la politique de collaboration définie pour chaque *SuperSession*, ces notifications d'événement peuvent conduire à exécuter des actions spécifiques dans ces applications.

Le niveau d'interaction entre les applications intégrées dépend évidemment de la nature des événements que les applications sont capables d'échanger et des actions qu'elles sont capables d'exécuter. Nous pouvons envisager trois cas principaux lors de l'intégration d'applications (en fonction de leur ouverture) :

(i) Les applications sont *open source* – L'intégration d'applications *open source* permet de réaliser le niveau d'intégration le plus élaboré, étant donné que tout événement (et toute action) peut être exporté (et exécutée). Néanmoins, la modification d'applications *open source*, pour obtenir le niveau d'intégration désiré, peut également conduire à des efforts de développement importants.

(ii) Les applications disposent d'une interface de programmation (API) – Une API permet à des modules logiciels extérieurs à l'application d'accéder aux ressources de celle-ci et de reproduire, par exemple, les mêmes actions que l'on peut effectuer à travers son interface utilisateur. De plus, des modules logiciels extérieurs peuvent également être avertis des actions exécutées à l'intérieur de l'application. Ainsi, l'intégration d'applications disposant d'APIs est directe. Bien évidemment, le niveau d'intégration dépend directement de l'API disponible.

(iii) Les applications ne disposent d'aucune API – Les applications ne disposant pas d'APIs sont contraintes à interagir uniquement via des actions *start* et *stop* de ces applications.

L'approche d'intégration définie par LEICA a pour cible le cas (ii). Nous sommes convaincus que les développeurs sont intéressés à créer des outils de collaboration qui puissent être utilisés soit en tant qu'outils indépendants, soit en tant qu'outils intégrés à d'autres applications (par l'intermédiaire d'une API la plus flexible possible), ceci, bien

entendu, afin de conquérir des parts de marché plus importantes pour leurs logiciels. A titre d'exemple, nous pouvons citer le logiciel *Skype*™ [Skype], un outil de communication sur l'Internet, très populaire aujourd'hui, et qui vient de publier son API.

Dans le but d'adapter des applications collaboratives et de les intégrer à LEICA, un *Wrapper* (ou empaqueteur) doit être utilisé. Le *wrapping* (ou empaquetage) est une technique d'adaptation traditionnelle où une application est emballée dans une nouvelle interface, pour la faire tourner dans un contexte différent. Cette technique représente une solution très appropriée pour notre environnement, surtout si nous partons du principe que les applications offrent des APIs. S'appuyant sur la description de l'API d'une application, LEICA est capable d'engendrer automatiquement un *Wrapper* pour cette application. Ainsi, nous adaptons l'interface de communication du *Wrapper* vis-à-vis de l'application, ce qui facilite le processus de couplage de ce *Wrapper*.

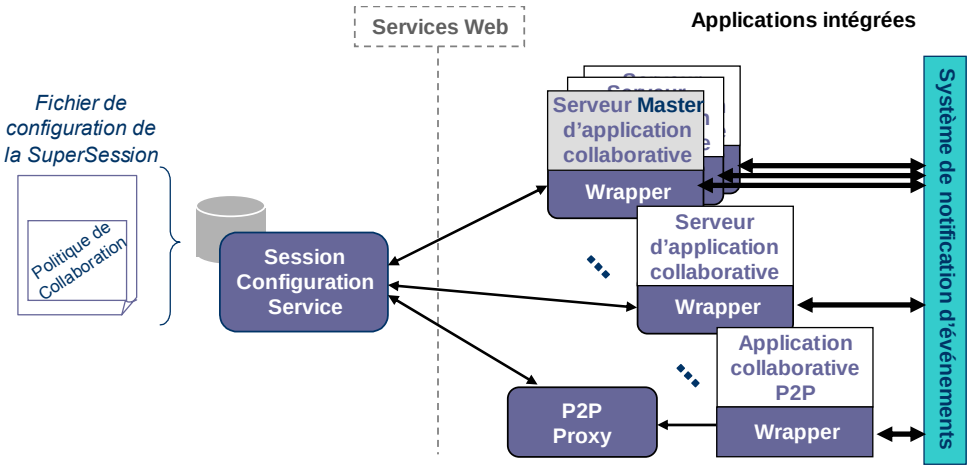

Figure IV.2 Le principe architectural d'intégration de LEICA

Comme cela est illustré dans la Figure IV.2, les *Wrappers* sont intégrés aux serveurs des applications client/serveur ou multiserveurs, et aux pairs des applications P2P (*peer-to-peer*). Dans le cas des serveurs, leurs *Wrappers* comprennent chacun une interface de services Web permettant à l'application d'interagir avec LEICA. L'application s'enregistre auprès de LEICA en tant qu'application intégrée et, à travers son interface de services Web, elle peut dialoguer avec le *Session Configuration Service* (ou service de configuration de session). Dans le cas des applications multiserveurs, un serveur *Master* (ou maître) est désigné pour enregistrer l'application et pour communiquer avec le *Session Configuration Service*.

Au contraire des applications client/serveur ou multiserveurs, les *Wrappers* des applications P2P ne présentent pas d'interface de services Web. Ce choix technique est lié au fait que les applications P2P sont en général exécutées de manière dynamique dans les machines hôtes lors de la connexion des utilisateurs. Par conséquent, elles ne peuvent pas s'exposer en tant que services Web. Nous nous sommes donc naturellement orientés vers la définition d'un *P2P Proxy* pour jouer le rôle de services Web pour les applications P2P. L'enregistrement auprès de LEICA, en tant qu'application intégrée, se fait donc par le biais de ce proxy. Et c'est lui qui doit interagir avec le *Session Configuration Service*.

Le *Session Configuration Service* se présente également comme un service Web. Ce service est utilisé pour créer et démarrer de nouvelles *SuperSessions*. Une *SuperSession* est une session collaborative intégrée gérant l'activité collaborative globale. Différentes applications collaboratives peuvent être utilisées en tant que support d'une *SuperSession*.

De même, dans le cadre d'une application collaborative, différentes sessions spécifiques peuvent exister. Une session spécifique est donc une session collaborative conventionnelle définie pour une application donnée (par exemple, une session de vidéoconférence ou de tableau blanc partagé).

Durant le processus de configuration d'une *SuperSession*, le *Session Configuration Service* contacte dynamiquement chaque application intégrée ainsi que le *P2P Proxy* pour leur demander :

(i) les données spécifiques nécessaires à la création de sessions spécifiques (par exemple, un outil de vidéoconférence peut avoir besoin d'une adresse multicast IP) ;

(ii) les types d'événement que l'application peut notifier et les types de requêtes d'action qu'elle peut recevoir (nous appelons cette information l'"API événements/actions"). Les APIs événements/actions seront en particulier nécessaires lors de la spécification de la politique de collaboration de la *SuperSession*.

Lors de la configuration d'une *SuperSession*, en plus de spécifier des informations générales de gestion, de choisir les applications (parmi l'ensemble de toutes les applications intégrées à LEICA) et de configurer les applications collaboratives à utiliser, on spécifie également la politique de collaboration associée à la *SuperSession*. Une politique de collaboration d'une *SuperSession* représente un ensemble de règles exprimées sous la forme d'un modèle conditions/actions. Ces règles définissent comment chaque application collaborative de la *SuperSession* doit réagir lorsque des événements en provenance d'autres applications de la *SuperSession* sont notifiés. En d'autres termes, les règles déterminent les actions que les applications intégrées doivent exécuter en réponse à des notifications d'événement (sous certaines conditions). La politique de collaboration spécifie ainsi le couplage souhaité entre les applications intégrées lors de l'exécution d'une *SuperSession*, et par conséquent, le comportement dynamique de la *SuperSession*.

Une fois qu'une *SuperSession* a été créée, configurée et son fichier de configuration enregistré (ce fichier est stocké dans une base de données locale au *Session Configuration Service*), le *Session Configuration Service* peut la démarrer. Les applications définies en tant que support de la *SuperSession* sont alors contactées (pour recevoir les informations concernant cette *SuperSession*), et elles sont finalement connectées au moyen d'un service de notification d'événements. Nous n'utiliserons donc plus de services Web à partir de ce moment, pour les raisons que nous avons indiquées dans la section II.4.4.

Au cours de la *SuperSession*, pendant que les applications collaboratives sont en train d'exécuter leurs sessions spécifiques, les *Wrappers* échangent des notifications d'événement en mode pair à pair. Les *Wrappers* sont également responsables de la gestion des règles de collaboration. Ainsi, lorsqu'un *Wrapper* reçoit une notification d'événement, il vérifie si cet événement sensibilise une règle politique concernant l'application collaborative à laquelle il est attaché (c'est-à-dire, une règle politique ayant des requêtes d'actions pour cette application). Dans l'affirmative, il envoie la (les) requête(s) d'action associée(s) à l'application.

Remarquons que LEICA n'a pas du tout pour objectif de prendre en compte des événements physiques de bas niveau (comme par exemple, des clics ou des déplacements de souris) ni des événements de synchronisation à fréquence élevée (comme par exemple la position courante d'objets qui se déplacent). LEICA a au contraire pour objectif de prendre en compte des événements significatifs qui véhiculent une sémantique applicative. En revenant, par exemple, sur notre scénario de *E-learning*, nous pouvons repérer des

événements tels que l'entrée d'un avatar dans une salle de cours du monde virtuel. Une règle politique pourrait ensuite associer la notification de cet événement (provenant du CVE) à des actions demandant à *CoLab* et à l'outil d'audioconférence de connecter l'utilisateur concerné à la session spécifique correspondant à cette salle de cours.

Figure IV.3 synthétise le cadre général d'intégration défini pour LEICA.

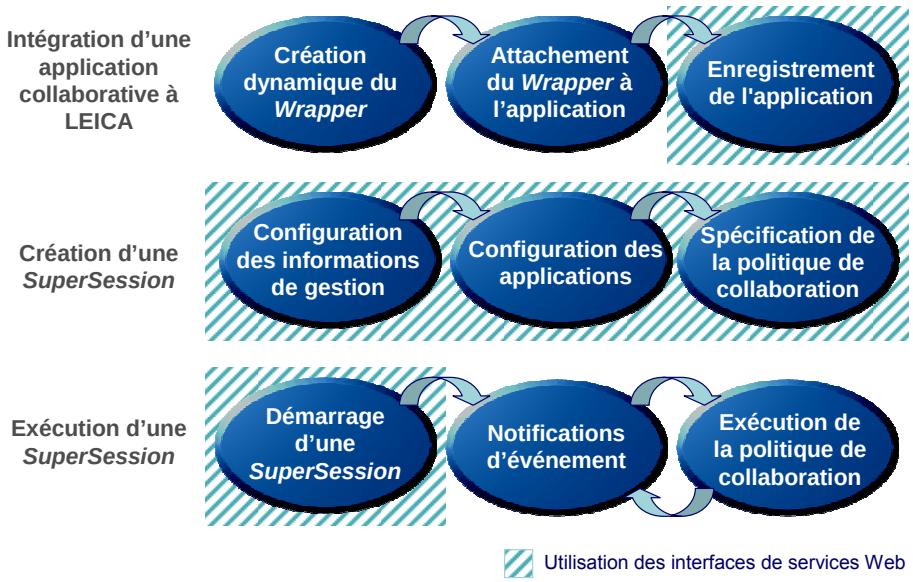

Figure IV.3 Le cadre général d'intégration de LEICA

IV.3. Description d'une *SuperSession*

Comme nous l'avons introduit précédemment, une activité collaborative se déroule dans le contexte d'une *SuperSession*. Dans le but de bien identifier et décrire tous les éléments composant une *SuperSession*, nous avons défini un modèle de session. LEICA s'appuie sur ce modèle afin de gérer l'état global d'une *SuperSession* de manière concise et cohérente. Une taxonomie bien définie des entités et de leurs attributs peut être également inférée de ce modèle.

De nombreux modèles ont été proposés dans le contexte des systèmes TCAO. Certains modèles [Ellis-94] [Dommel-00] visent à définir un cadre conceptuel pour décrire et caractériser les composants des systèmes TCAO, ce qui permet également de définir des paramètres pour classer et comparer ces systèmes. D'autres modèles se focalisent plus particulièrement sur l'aspect "intégration". Dans [Teege-96] et [Farias-00] les auteurs proposent des modèles qui reposent sur une notion assez générale d'"activité" dans le but de structurer les éléments d'un environnement collaborative intégré. Ces différents modèles ne présentent cependant pas le niveau de détail que nous souhaitons pour caractériser une *SuperSession*. Néanmoins, nous nous sommes appuyés sur plusieurs concepts introduits dans ces modèles pour définir notre modèle de *SuperSession*.

IV.3.1. Modèle de *SuperSession*

La *SuperSession SS* est l'abstraction de base du modèle, qui comprend, selon une structure hiérarchique, toutes les entités actives d'une session globale exécutée dans LEICA. La Figure IV.4 schématise l'organisation de ces entités.

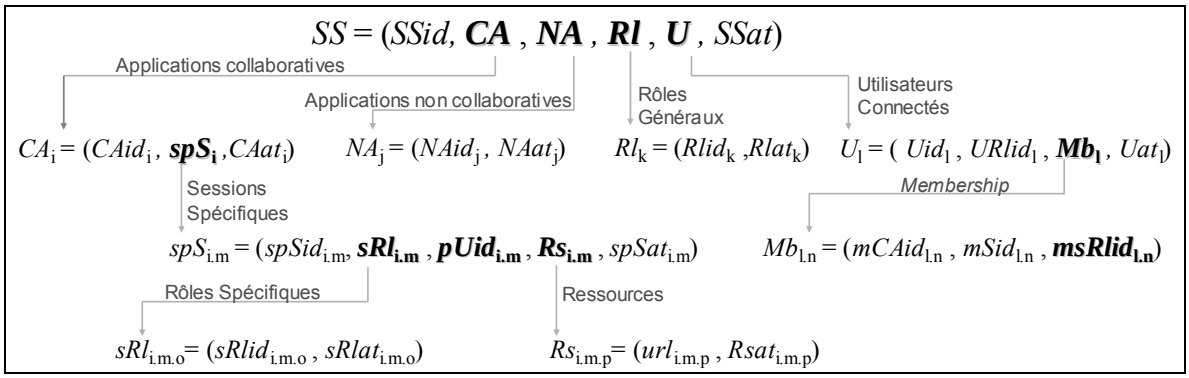

Figure IV.4 Synthèse du modèle de la SuperSession

Une *SuperSession* représente une activité collaborative comprenant plusieurs applications, un groupe d'utilisateurs et de rôles généraux associés à ces utilisateurs. Plus formellement, une *SuperSession* SS est un n-uplet :

$$SS = (SSid, \mathbf{CA}, \mathbf{NA}, \mathbf{Rl}, \mathbf{U}, SSat)$$

où *SSid* est l'identificateur unique d'une *SuperSession* ; $\mathbf{CA}=\{CA_i\}_{i\leq\alpha}$ est un ensemble fini d'applications collaboratives ; $\mathbf{NA}=\{NA_j\}_{j\leq\beta}$ [1] est un ensemble fini d'applications non collaboratives ; $\mathbf{Rl}=\{Rl_k\}_{k\leq\gamma}$ est l'ensemble fini de rôles généraux ; $\mathbf{U}=\{U_l\}_{l\leq\delta}$ est l'ensemble fini d'utilisateurs connectés ; *SSat* est une liste d'attributs caractérisant une *SuperSession*. Ces attributs décrivent des informations telles que le contexte de la session (nom, propos, *etc.*), la planification (*scheduling*) de la session (si son début est programmé ou pas, sa durée, *etc.*), le type d'accessibilité (ouverte ou fermée) et le nombre maximum d'utilisateurs autorisés.

IV.3.1.1. Applications collaboratives

Généralement, une activité collaborative globale se décompose en plusieurs tâches. Pour supporter ces tâches, différentes applications collaboratives peuvent être employées. Par ailleurs, chaque application contrôle ses propres sessions collaboratives, que nous appelons ici "sessions spécifiques". Formellement, une application collaborative est un n-uplet :

$$CA_i = (CAid_i, \mathbf{spS}_i, CAat_i)$$

où $CAid_i$ est son identificateur unique ; $\mathbf{spS}_i=\{spS_{i.m}\}_{m\leq\varepsilon}$ est un ensemble fini de sessions spécifiques ; et $CAat_i$ est une liste d'attributs caractérisant une application collaborative. Ces attributs fournissent des informations sur l'application : son nom, son type, si elle supporte ou pas la notion de rôle, son architecture (client/serveur, multiserveurs, P2P) et le type d'application utilisateur (*stand-alone* ou *web-based*).

IV.3.1.2. Applications non collaboratives

Même si l'environnement LEICA a été prévu pour l'intégration d'applications collaboratives, son schéma d'intégration permet également l'intégration d'applications non collaboratives. Prenons comme exemple le scénario de *E-learning* présenté dans la section IV.2.1.3. Supposons que nous souhaitions représenter dans chaque salle de cours virtuelle

[1] $(CA \cup NA) \subseteq \Delta$, où Δ représente l'ensemble fini comprenant toutes les applications intégrées à LEICA.

le contexte de l'activité de navigation Web en cours. Pour cela, il suffit d'ajouter dans la salle un objet 3D représentant la page Web actuellement visitée par le groupe d'utilisateurs présents dans cette salle. Dans ce cas une application capable de générer dynamiquement des objets 3D à partir de documents HTML devra être employée.

Formellement, une application non collaborative est un couple :

$$NA_j = (NAid_j , NAat_j)$$

où $NAid_j$ est son identificateur unique et $NAat_j$ représente les attributs caractérisant cette application (nom et propos).

IV.3.1.3. Rôles généraux

Chaque utilisateur connecté à une *SuperSession* doit être associé à un rôle général. Le concept de rôle général est utilisé pour grouper les utilisateurs ayant le même ensemble de responsabilités et privilèges dans une *SuperSession*. Formellement, un rôle général est un couple :

$$Rl_k = (Rlid_k , Rlat_k)$$

où $Rlid_k$ est son identificateur unique et $Rlat_k$ est une liste d'attributs caractérisant ce rôle général. Cette liste donne des détails sur le rôle tels que sa description, sa méthode d'attribution et ses droits d'administration. La méthode d'attribution définit comment un rôle est associé à un utilisateur : cela peut se faire de manière (i) statique (on prédéfinit une liste avec la composition du groupe) ou (ii) par le choix de l'utilisateur (protégé par mot de passe ou pas). Concernant les droits d'administration, un rôle général avec le statut de *chairman* permet aux utilisateurs d'agir en tant qu'administrateur ou modérateur de la *SuperSession*, pouvant, par exemple, choisir le moment de l'arrêter, ou même de bannir un autre utilisateur. La politique d'attribution du statut de *chairman* est également définie : si tous les utilisateurs connectés à ce rôle peuvent être des *chairman*, ou si seulement les *n* premiers ou les *n* derniers à se connecter peuvent le devenir.

IV.3.1.4. Utilisateurs connectés

Cette entité représente tous les utilisateurs qui ont rejoint une même *SuperSession*. Comme nous l'avons dit précédemment, chaque utilisateur est associé à un rôle général. Une fois connectés à une *SuperSession*, les utilisateurs peuvent participer simultanément à différentes sessions spécifiques définies pour les applications collaboratives utilisées dans cette *SuperSession*. Nous appelons ces participations "relations de *Membership*". Formellement, un utilisateur connecté est un n-uplet :

$$U_l = (Uid_l , URlid_l , \boldsymbol{Mb}_l , Uat_l)$$

où Uid_l est son identificateur unique ; $URlid_l$ est l'identificateur d'un des rôles généraux ; $\boldsymbol{Mb}_l=\{Mb_{l.n}\}_{n\leq\phi}$ est l'ensemble fini des relations de *Membership* courantes ; Uat_l est une liste d'attributs caractérisant l'utilisateur. Ces attributs définissent des informations personnelles de l'utilisateur (nom et email) ainsi que des informations techniques (*e.g.* son adresse IP).

Chaque relation de *Membership* est caractérisée par un triplet :

$$Mb_{l.n} = (mCAid_{l.n} , mSid_{l.n} , \boldsymbol{msRlid}_{l.n})$$

où $mCAid_{l.n}$ identifie l'application collaborative dans laquelle cette relation est établie ; $mSid_{l.n}$ est l'identificateur d'une session spécifique et $\boldsymbol{msRlid}_{l.n}=\{msRlid_{l.n.q}\}_{q\leq\mu}$ est l'ensemble fini d'identificateurs de rôles spécifiques associés à cet utilisateur dans la session spécifique identifiée par $mSid_{l.n}$. Autrement dit, chaque relation de *Membership* indique la participation d'un utilisateur à une session spécifique, et identifie le(s) rôle(s) spécifique(s) associé(s) à cet utilisateur.

IV.3.1.5. Sessions spécifiques

Une session spécifique représente une session collaborative conventionnelle au sein d'une application collaborative (par exemple, une session d'audioconférence ou une session de navigation Web coopérative). La présence de cet élément dans notre modèle n'a pas pour objectif de décrire tous les aspects internes d'une session collaborative spécifique à une application. Rappelons que LEICA n'a pas pour but de gérer les applications collaboratives. En réalité, cela nous empêcherait de concevoir LEICA en tant qu'environnement d'intégration faiblement couplé. Ainsi, une session spécifique caractérise plutôt les éléments communs à différentes applications collaboratives tels que les rôles spécifiques définis pour cette session (si l'application définit ce concept), les utilisateurs participants à cette session, ainsi que les ressources partagées au cours de cette session spécifique. Remarquons qu'une application collaborative n'est pas sensée définir explicitement cette abstraction de session spécifique. Néanmoins, elle doit, même de façon implicite, supporter la notion de groupe d'utilisateurs appartenant à une séance de travail commune.

Formellement, une session spécifique est un n-uplet :

$$spS_{i.m} = (spSid_{i.m}, \boldsymbol{sRl}_{i.m} , \boldsymbol{pUid}_{i.m} , \boldsymbol{Rs}_{i.m} , spSat_{i.m})$$

où $spSid_{i.m}$ est son identificateur unique ; $\boldsymbol{sRl}_{i.m}=\{sRl_{i.m.o}\}_{o\leq\eta}$ est un ensemble fini de rôles spécifiques ; $\boldsymbol{pUid}_{i.m}=\{pU_{i.m.p}\}_{p\leq\iota}$ est un ensemble fini d'identificateurs d'utilisateurs ; $\boldsymbol{Rs}_{i.m}=\{Rs_{i.m.p}\}_{p\leq\varphi}$ est un ensemble fini de ressources ; et $spSat_{i.m}$ est une liste d'attributs caractérisant la session spécifique. Ces attributs décrivent des information telles que la planification, l'accessibilité (ouverte ou fermée) et la méthode d'attribution des rôles spécifiques : (i) statique (liste d'utilisateurs) ; (ii) automatique (basée sur le rôle général de l'utilisateur) ; ou (iii) au choix de l'utilisateur (protégée ou pas par mot de passe).

Dans le cas où l'application collaborative offre la notion de rôle, nous avons $|\boldsymbol{sRl}_{i.m}|>0$. Un rôle spécifique est un couple :

$$sRl_{i.m.o} = (sRlid_{i.m.o} , sRlat_{i.m.o})$$

où $sRlid_{i.m.o}$ est son identificateur unique et $sRlat_{i.m.o}$ une liste d'attributs caractérisant le rôle spécifique. Ces attributs présentent la description du rôle, le nombre maximum d'utilisateurs, et les informations concernant sa composition (*membership*).

Une ressource est également formalisée par un couple :

$$Rs_{i.m.p}= (uri_{i.m.p} , Rsat_{i.m.p})$$

où $uri_{i.m.p}$ est son URI (de l'anglais *Uniform Resource Identifier*, soit littéralement "identifiant uniforme de ressource") et $Rsat_{i.m.p}$ une liste d'attributs caractérisant la ressource. Le propos de cet élément est simplement de permettre l'implantation d'un mécanisme de contrôle d'accès concurrent entre applications. Par conséquent, nous ne représentons que les ressource dont l'existence n'est pas limitée au contexte d'une application (c'est-à-dire, plutôt des fichiers et des périphériques que des objets virtuels). La

liste d'attributs comprend uniquement : le type de la ressource et le type d'accès effectué par l'application (lecture et/ou écriture).

IV.3.2. Illustration d'une *SuperSession*

Afin d'illustrer notre modèle de *SuperSession*, nous allons reprendre le scénario d'intégration de *E-learning* présenté dans la section IV.2.1.3.

Dans ce scénario nous identifions cinq applications collaboratives : le CVE (CA_1), le Chat (CA_2), l'outil *CoLab* de navigation Web coopérative (CA_3), l'outil d'audioconférence (CA_4) et le tableau blanc partagé (CA_5). Le CVE exécute une seule session spécifique ($spS_{1.1}$) qui correspond au monde virtuel représentant le bâtiment 3D de l'école. Le Chat exécute également une seule session spécifique ($spS_{2.1}$) concernant la session de Chat du hall d'entrée. Nous avons de plus une session spécifique ($spS_{5.1}$) de tableau blanc partagé (associée à la salle des enseignants). *CoLab* et l'outil d'audioconférence doivent exécuter deux sessions spécifiques chacun ($spS_{3.1}$ et $spS_{3.2}$ dans *CoLab* ; $spS_{4.1}$ et $spS_{4.2}$ dans l'audioconférence), de façon à mettre en place les deux cours prévus dans ce scénario.

Supposons qu'un document de type image soit accédé en même temps pendant l'exécution de deux sessions spécifiques : dans la session spécifique du tableau blanc partagé les utilisateurs sont en train d'éditer ce document ($Rs_{5.1.1}$), tandis que dans le monde virtuel, plus précisément dans la salle des enseignants, l'image définie dans ce document est plaquée sur un objet représentant un tableau blanc ($Rs_{1.1.1}$).

Nous identifions également dans ce scénario les rôles généraux représentant les trois types d'utilisateurs qui peuvent se connecter: "Élève" (Rl_1), "Moniteur" (Rl_2) et "Professeur" (Rl_3). De plus, dans la section IV.3.1.2 nous avons étendu ce scénario en prévoyant une application pour la génération automatique d'objets 3D à partir de documents HTML. Nous appelons cette application non collaborative "html_to_3D" (NA_1).

La Figure IV.5 illustre cette *SuperSession* (que nous avons appelée de "E-learning") et ses entités. Nous avons choisi la notation SS^t pour représenter un instantané d'une *SuperSession* (et récursivement de ses éléments). En fait, si d'un coté certaines entités d'une *SuperSession*, que nous appelons "entités statiques", restent inchangées lors de leur exécution (par exemple, les applications et les rôles généraux), d'autres entités, appelées "entités dynamiques", peuvent bien évidement évoluer dans le temps (*e.g.* les utilisateurs connectés, les ressources manipulées, *etc.*).

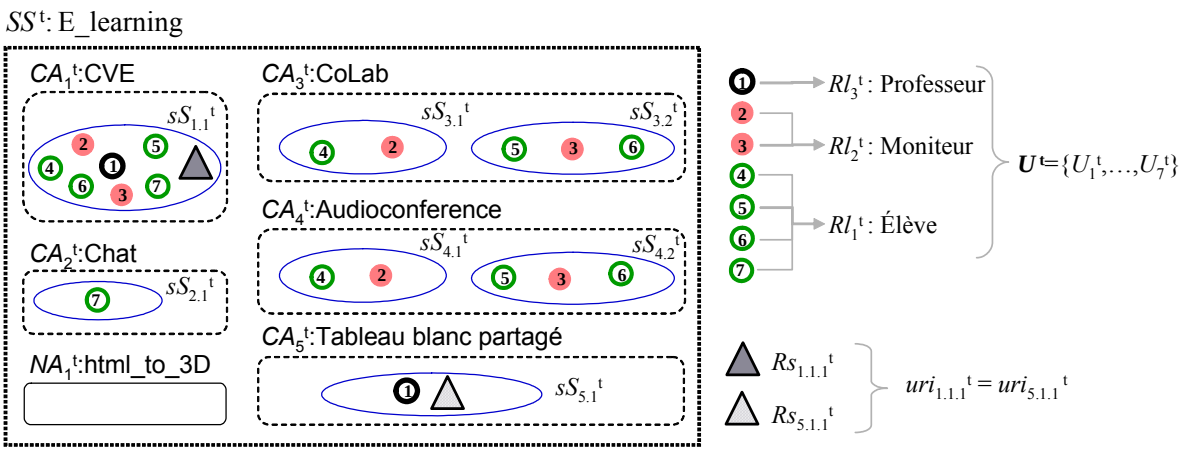

Figure IV.5 Illustration du modèle de la SuperSession de E-learning

IV.4. Création d'une *SuperSession*

Créer une *SuperSession* consiste à définir les données qui caractérisent cette session et son comportement. Nous devons tout d'abord fournir un ensemble d'informations de gestion, sélectionner et configurer les applications (parmi l'ensemble des applications intégrées à LEICA) qui seront utilisées en support de la *SuperSession*. Nous devons ensuite caractériser le comportement de la *SuperSession* par la spécification d'une politique de collaboration qui définit comment les différents composants de la *SuperSession* sont coordonnés.

IV.4.1. Configuration des informations de gestion

Les informations de gestion se partagent en deux groupes : (i) *GSMinfo* (de l'anglais *General Session Management information*, soit littéralement "Informations générale de gestion de session") ; et (ii) *IAinfo* (de l'anglais *Integrated Applications information*, soit littéralement "Informations d'applications intégrées").

IV.4.1.1. GSMinfo

Le *GSMinfo* décrit les informations de contexte d'une *SuperSession*, sa planification, et la définition des rôles généraux. Le tableau illustré dans la Figure IV.6 résume les champs de configuration concernant le *GSMinfo*.

Contexte	
Name, Description	String
Planification	
Starting	*CCYY-MM-DD*T*hh*:*mm*:*ss* | (programmée) *hh*:*mm*:*ss*-[Su][Mo][Tu][We][Th][Fr][Sa] (programmée) null (démarrée manuellement)
Duration	*CCYY-MM-DD*T*hh*:*mm*:*ss* | (programmée) *hh*:*mm*:*ss* | P*y*Y*m*M*d*DT*h*H*m*M*s*S (programmée) null (arrêtée manuellement ou suite à la politique de collaboration)
Liste de rôles généraux	
Role ID, Description	String
Membership	'['+(user-identifier,passwd)']' | (liste avec la composition du groupe) password | (choix de l'utilisateur protégé par mot de passe) null (choix de l'utilisateur)
Max. numb. simultaneous users	Integer
Administration assignment	null | all | *n* first | *n* last (n > 0)

Figure IV.6 Les champs de configuration du GSMinfo

IV.4.1.2. IAinfo

Le *IAinfo* définit quelles seront les applications utilisées dans la *SuperSession*. Dans le cas d'applications non collaboratives, on sélectionne simplement les applications et aucune autre configuration n'est nécessaire. Dans le cas d'applications collaboratives, une fois ces applications sélectionnées, nous devons fournir des informations pour configurer les sessions spécifiques associées à ces applications collaboratives.

Le tableau illustré dans la Figure IV.7 résume les champs de configuration communs aux sessions spécifiques. Ces informations doivent être fournies pour toutes les sessions. De plus, des informations propres à chaque application collaborative doivent également être données. Rappelons que les applications collaboratives informent LEICA des types d'information dont elle a besoin pour la création d'une session spécifique.

Contexte		
Session identifier	String	
Planification		
Starting	*CCYY-MM-DD*T*hh*:*mm*:*ss* | (programmée, comme la SuperSession) *hh*:*mm*:*ss*-[Su][Mo][Tu][We][Th][Fr][Sa] | (programmée, comme la SuperSession) null (démarrée manuellement ou suite à la politique de collaboration)	
Duration	*CCYY-MM-DD*T*hh*:*mm*:*ss* | (programmée) *hh*:*mm*:*ss* | P*y*Y*m*M*d*DT*h*H*m*M*s*S | (programmée) null (arrêtée manuellement ou suite à la politique de collaboration)	
Rôles spécifiques		
Role association:	"none" | "static" | "automatic" | "anonymous"	
Si *Role association = "none"* (l'application ne définit pas la notion rôle)	*Membership*	'['+(user-identifier,passwd)']'| (liste statique d'utilisateurs, session fermée) password | (session fermée) null (session ouverte)
Liste de rôles spécifiques		
Si *Role association = "static"* (liste statique)	Role identifier, Description	String
	Membership	'['+(user-identifier,passwd)']' (liste d'utilisateurs)
Si *Role association = "automatic"* (automatique, selon le rôle général de l'utilisateur)	Role identifier, Description, Password	String
	Function:	'['+(general-role)']' (liste de rôles généraux)
Si *Role association = "anonymous"* (choix de l'utilisateur)	Role identifier, Description, Password	String
Max_sim__users (nombre max. d'utilisateurs)		Integer

Figure IV.7 Les champs communs de configuration des sessions spécifiques

IV.4.2. Spécification des politiques de collaboration

L'association de politiques de collaboration à des *SuperSessions* s'avère un élément clé de notre environnement. C'est la spécification de ces politiques de collaboration qui permet de donner une sémantique à l'intégration, en définissant le comportement souhaité des applications collaboratives au cours des *SuperSessions*.

Les politiques de collaboration correspondent à un ensemble de règles qui permettent d'associer des notifications d'événement à des requêtes d'exécution d'action, sous réserve que certaines conditions explicites soient satisfaites. Ces règles ont pour objectif de lier (tout en offrant un certain niveau de coordination) les activités collaboratives en exécution au sein de chaque application utilisée en support d'une *SuperSession*.

Il existe plusieurs langages de spécification de politiques, notamment si nous regardons le domaine de la sécurité, où les exigences précises quant au niveau de sécurité

voulu pour un système particulier sont généralement exprimées à travers la politique de sécurité de ce système.

Nous avons par exemple le langage *PDL* (*Policy Description Language* [Lobo-99]), originalement développé par *Bell Labs*. La construction de base des règles en PDL est : "*event causes action if condition*". A chaque *event* sont attachés différents attributs. Les *actions* sont des commandes du système et les *conditions* des fonctions booléennes. Les deux types de fonctions prennent comme paramètres des attributs. Nous trouvons un certain niveau de modularité dans ce langage qui rend difficile la spécification de règles plus complexes que le simple modèle événement-condition-action.

Un autre exemple est *Ponder* [Damianou-01], un langage assez général, déclaratif et orienté objet. Différents types de règles peuvent être définis: autorisation, interdiction, délégation, contention, et obligation. Ce dernier type est en particulier intéressant pour LEICA vu qu'il permet de spécifier les actions à entreprendre obligatoirement lorsque certains événements surviennent. *Ponder* est considéré un langage complet, très riche mais qui nous a semblé parfois assez confus.

Dans le domaine du TCAO, d'autres formalismes ont été employés pour permettre la coordination des acteurs (utilisateurs, artefacts, outils, *etc.*) au sein d'une session collaborative. Par exemple, dans [Rodríguez-03], l'auteur présent le MDSC, un modèle de session synchrone basé sur des diagrammes de coordination. Le MSDC permet de pré-programmer et de représenter des sessions structurées complexes (*i.e.* composées par différentes applications collaboratives). Un concepteur de session peut définir des diagrammes de coordination représentés par des graphes étiquetés dont les noeuds représentent les utilisateurs et les flèches représentent les échanges de données entre utilisateurs (par application). De cette façon il est possible de coordonner la façon dont chaque utilisateur doit utiliser chaque application collaborative ainsi que leurs interactions (et leurs rôles) pour chaque application.

Dans le cas de LEICA, notre but n'est pas de gérer la coordination des acteurs à l'intérieur des applications collaboratives. Nous traitons en effet chaque application collaborative en tant qu'entité indépendante responsable de gérer ses propres sessions spécifiques. Nous pourrions néanmoins nous appuyer sur des formalismes qui, dans certains cas, offrent un support pour la coordination au moyen de politiques et de règles. Mais, dans la plupart des cas, la spécification de politiques vise plus à restreindre le comportement des systèmes qu'à définir un comportement. *Intermezzo* [Edwards-96], par exemple, prévoit la spécification de règles de coordination qui définissent uniquement les rôles des utilisateurs et les droits accordés à ces rôles. D'autres travaux ont prévu la définition de "règles d'obligation" (comme le fait *Ponder*). Par exemple, dans le cadre formel défini dans [Molina-Espinosa-04], l'un des opérateurs utilisés dans la définition des règles de coordination, en l'occurrence l'opérateur *ImEnable(α,β)*, impose que l'exécution de l'action α soit immédiatement suivie de l'action β.

Pour LEICA, nous envisageons un mécanisme de spécification de règles qui soit beaucoup plus intuitif pour les utilisateurs (ceux qui définissent les *SuperSessions*), qui leur épargne l'apprentissage d'un formalisme parfois complexe, mais qui leur permette également de disposer d'une sémantique de collaboration qui soit plus riche qu'un simple modèle événement-condition-action. Nous avons ainsi fait le choix d'une méthode graphique pour la spécification des règles de coordination. Par le biais d'un simple éditeur graphique, nous pourrons définir des règles en composant des éléments graphiques appelés *policy widgets*. Ces *widgets* permettent de définir des règles simples, qui associent la notification d'un événement à une requête d'exécution d'une action, ou des règles plus

complexes qui associent la notification de *n* événements à *m* requêtes d'exécution d'action. Pour composer ces règles, nous devons connaître l'ensemble des types d'événement que les applications sont capables de notifier, ainsi que toutes les requêtes d'exécution d'action qu'elles peuvent accepter. Nous voyons par conséquent l'importance des APIs événements/actions que nous avons brièvement introduites dans la section IV.2.2.

Une fois que les règles de collaboration ont été définies graphiquement, une spécification textuelle des règles est engendrée et attachée au fichier de configuration de la *SuperSession*. La sémantique de ces règles de collaboration est définie par traduction en réseaux de Petri, comme nous le verrons dans le paragraphe IV.4.2.4. Cette sémantique est mise en œuvre par un sous-module spécifique du *Wrapper* qui est en charge d'exécuter les règles.

IV.4.2.1. Types de notification d'événement et de requêtes d'exécution d'action

Lorsqu'une application définit son API événements/actions, elle doit spécifier pour chaque événement et pour chaque action :

(i) le type – il s'agit d'un nom unique dans le groupe d'événements ou d'actions de cette application (différentes applications peuvent avoir des types d'événements/actions ayant les mêmes noms, ce qui ne signifie pas nécessairement qu'ils aient la même sémantique) ;

(ii) la liste de paramètres – chaque paramètre a un nom et un type (ce dernier indique s'il s'agit d'une valeur simple ou d'une liste de valeurs, ainsi que le type à proprement dit de ces valeurs).

Un paramètre spécial est ajouté automatiquement à tous les événements et à toutes les actions d'une API. Ce paramètre, appelé "spSid", sert à identifier la session spécifique d'où provient un événement, ou la session spécifique où une requête d'action doit être exécutée. Dans l'Annexe A (section A.2.2), nous précisons la syntaxe choisie pour la spécification de l'API événements/actions d'une application.

Au delà de l'API événements/actions définie pour chaque application intégrée à LEICA, nous disposons d'un ensemble prédéfini d'événements et d'actions que nous appelons "API de gestion". La première partie de cette API concerne les événements de gestion qu'une application collaborative (les applications non collaboratives ne sont pas concernées) doit être capable de notifier, ainsi que les requêtes d'exécution d'action qui doivent être traitées. Ces notifications d'événement correspondent aux types[1] :

- CONNECT – pour notifier la connexion d'un utilisateur dans une session spécifique (il comprend les paramètres "spSid", identifiant la session spécifique, "Uid", identifiant l'utilisateur, et "sRlid", identifiant son/ses rôle(s) spécifique(s)) ;
- DISCONNECT – pour notifier la déconnexion d'un utilisateur d'une session spécifique (il comprend uniquement les paramètres "spSid" et"Uid") ;
- RESOURCE [2] – pour notifier l'accès à une ressource dans une session spécifique (il comprend les paramètres "spSid", "uri", indiquant l'URI de la ressource , "type", indiquant le type de ressource, et "access", indiquant le type d'accès à la ressource).

[1] Voir la section A.2.3 de l'Annexe A pour plus de détails sur la syntaxe des événements et des actions de l'API de gestion.

[2] L'événement "RESOURCE" est optionnel.

Les requêtes d'exécution d'action de l'API de gestion pour les applications collaboratives correspondent également aux types CONNECT, DISCONNECT et RESOURCE ; le deux premières concernent respectivement la connexion et la déconnexion d'utilisateurs dans des sessions spécifiques. Néanmoins, nous avons pris en compte que certaines applications collaboratives puissent ne pas avoir la capacité de forcer la connexion ni la déconnexion des utilisateurs, surtout dans le cas d'applications client/serveur ou multiserveurs, où ces fonctionnalités sont normalement assurées par les applications clientes (à l'initiative de l'utilisateur lui-même), et non pas par les serveurs. Pour ne pas contraindre l'intégration de ces applications, cette partie de l'API de gestion reste ouverte : chaque application définira les paramètres pour les actions CONNECT et DISCONNECT, et indiquera si elle est capable ou pas de traiter ces requêtes. Dans les cas où les applications ne pourront pas les traiter, LEICA s'en chargera. Quant à la requête d'exécution d'action RESOURCE, elle est optionnelle et représente une requête à l'application pour qu'elle change le type d'accès à une ressource, ou en interrompe l'accès. Cette requête comprend les paramètres "spSid", "uri", et "access".

La deuxième partie de l'API de gestion concerne les événements et actions de gestion relatifs à LEICA. Ces événements et actions ne sont donc pas traités par les applications intégrées, mais par l'environnement lui-même. Concernant les notifications d'événement, nous avons :

- CONNECT – pour notifier la connexion d'un utilisateur dans une *SuperSession* (il comprend les paramètres "Uid", identifiant l'utilisateur, "name", indiquant son nom d'usage, "Rlid", identifiant le rôle général de l'utilisateur, et "ip", indiquant son adresse IP) ;
- DISCONNECT – pour notifier la déconnexion d'un utilisateur d'une *SuperSession* (il comprend le paramètre "Uid", identifiant l'utilisateur).

Nous avons un seul type de requête d'action pouvant être traité par LEICA :

- DISCONNECT – pour forcer la déconnexion d'un utilisateur d'une *SuperSession* (il comprend le paramètre "Uid", identifiant l'utilisateur).

IV.4.2.2. La définition des règles de collaboration : les *policy widgets*

La Figure IV.8 illustre les six *widgets* utilisés pour la définition des règles de collaboration. Ces *widgets* peuvent être raccordés par leurs points de connexion respectifs ou couplés par leurs interfaces de couplage. Chaque règle est ainsi le résultat de la composition de différents *widgets*. Les principes de base pour la composition des *widgets* sont définis ci-dessous :

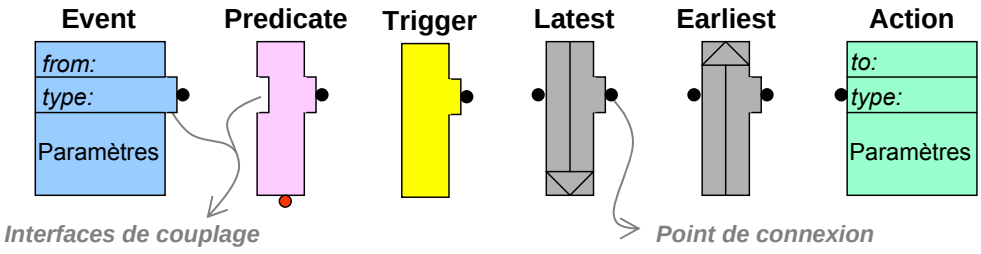

Figure IV.8 Illustration des policy widgets

(i) les règles politiques sont lues de la gauche vers la droite ;

(ii) seuls les *widgets* n'ayant pas de point de connexion ni d'interface de couplage à leur gauche (*i.e. Event* et *Trigger*) peuvent apparaître à l'extrémité gauche d'une règle ;
(iii) seuls les *widgets* n'ayant pas de point de connexion ni d'interface de couplage à leur droite (*i.e. Action*) peuvent apparaître à l'extrémité droite d'une règle ;
(iv) chaque règle doit être composée d'au moins un *widget* de type *Event* ou *Trigger* et d'un *widget* de type *Action*.

Au cours de l'exécution d'une *SuperSession*, les règles définissant sa politique de collaboration sont exécutées en parallèle. Une règle en cours d'exécution est appelée une règle active. Lorsque les *widgets* composant une règle sont sensibilisés, la règle est sensibilisée et peut donc être tirée.

Le *widget Event* représente une notification d'événement. Chaque *Event* est associé à une application collaborative (par le champ *from*) et à un type (par le champ *type*). Dans la section "Paramètres" nous trouvons la liste de paramètres définie pour un type donné d'événement d'une application. Lors de l'utilisation d'un *widget Event* dans une règle politique, nous pouvons spécifier des motifs (*matching patterns*) jouant le rôle de filtres qui sont appliqués aux valeurs des paramètres de l'événement respectif. Ainsi, un *widget Event* n'est sensibilisé que lorsque l'événement spécifié est notifié et que les valeurs des paramètres associés à cette notification correspondent aux motifs spécifiés dans le *widget*. Le lecteur trouvera dans l'Annexe B (section B.3.1) les détails de la syntaxe définie pour la spécification des motifs des paramètres du *widget Event*.

Le *widget Action* représente une requête d'exécution d'action. Chaque *Action* est associée à une application collaborative (par le champ *to*) et à un type (par le champ *type*). Dans la section "Paramètres" nous trouvons la liste de paramètres définie pour un type donné de requête d'exécution d'action d'une application. Chaque fois qu'un *widget Action* est utilisé dans une règle, nous devons spécifier les valeurs de tous les paramètres définis pour le type respectif de requête d'exécution d'action. Ces valeurs peuvent être des constantes, mais il est également possible de faire référence (i) à des paramètres provenant des notifications d'événement qui ont déclenché la sensibilisation de la règle politique, et (ii) à des composantes de l'état global courant de la *SuperSession*. Dans l'Annexe B (section B.3.3) nous trouvons les détails de la syntaxe définie pour la spécification des valeurs des paramètres du *widget Action*.

La Figure IV.9 illustre une règle politique simple qui associe directement un *Event*[1] à une *Action*. Pendant l'exécution de cette règle, lorsqu'un événement de type "T1" est notifié par l'application "CA1", le *widget Event* ne sera sensibilisé que si les paramètres "a" et "b" contenus dans cette notification ont des valeurs conformes aux motifs définis pour ces paramètres. Dans le cas affirmatif, la règle sera sensibilisée et donc tirée – une requête d'exécution d'action de type "T1" sera envoyée à l'application "CA2".

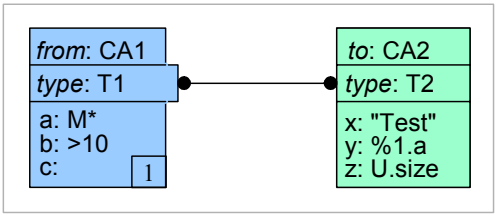

Figure IV.9 Une règle politique simple

[1] Par souci de simplicité, nous n'allons pas afficher le paramètre "spSid" (défini par défaut pour tout événement) dans le *widget Event*.

Toujours dans la Figure IV.9, nous notons que le paramètre "z" du *widget Action* est associé à une composante de l'état global de la *SuperSession*. Nous indiquons ainsi que la valeur de ce paramètre correspond au nombre d'utilisateurs actuellement connectés à la *SuperSession*. Remarquons également que le texte "%1.a" est associé au paramètre "y" du *widget Action*. Ceci indique que nous attribuons à ce paramètre la valeur du paramètre "a" de l'*Event* portant l'identificateur "1" (en fait le seul *Event* de la règle). Nous pouvons remarquer ici une caractéristique de la spécification des règles : lorsqu'un *widget Event* est utilisé dans une règle, il est automatiquement associé à un identificateur (numérique) unique. Nous avons fait ce choix pour permettre de référencer les paramètres de chaque *Event*. Dans l'Annexe B (section B.3) nous présentons les détails de la syntaxe définie pour les références à des paramètres et à des composantes de l'état global de la *SuperSession*.

Le *Predicate* est un *widget* qui permet de définir une condition (sous la forme d'un prédicat) pour sensibiliser une règle politique. Un *Predicate* peut être couplé à tous les autres *widgets* à l'exception du *widget Action*. Ainsi, le prédicat défini n'est évalué que lorsque cet autre *widget* est sensibilisé.

Dans l'exemple "a" de la Figure IV.10, nous avons un *Predicate* directement associé à un *Event*. Dans ce cas, pendant l'exécution de la règle, à chaque fois que l'événement respectif "T1" est notifié par "CA1" (et que son paramètre "a" contient la valeur "chair"), l'*Event* est sensibilisé et le prédicat peut donc être évalué. Supposons que nous ayons la séquence de notifications d'événements illustrée sur l'axe temporel présenté au dessous de la règle. En t_3, l'événement est notifié, et vu que le prédicat est vrai, la règle est sensibilisée et tirée. Ainsi, une requête d'exécution d'action de type "T3" est envoyée à "CA2". Nous pouvons remarquer, dans cet exemple, une particularité du *widget Predicate*, à savoir son point de connexion alternatif (le point positionné au dessous du composant graphique). Ce point de connexion peut être utilisé sous réserve que le *Predicate* soit placé juste avant le(s) *Action*(s). Il permet d'exprimer un comportement alternatif à la règle politique dans le cas où le prédicat est évalué à faux. Ainsi, toujours dans l'exemple "a", nous avons en t_1 la notification de l'événement "T1" (en supposant "a=chair") et, puisque à cet instant là le prédicat est faux, la règle est également sensibilisée et tirée. Par contre c'est la requête correspondante à l'*Action* placée en bas de la figure qui est envoyée à "CA2".

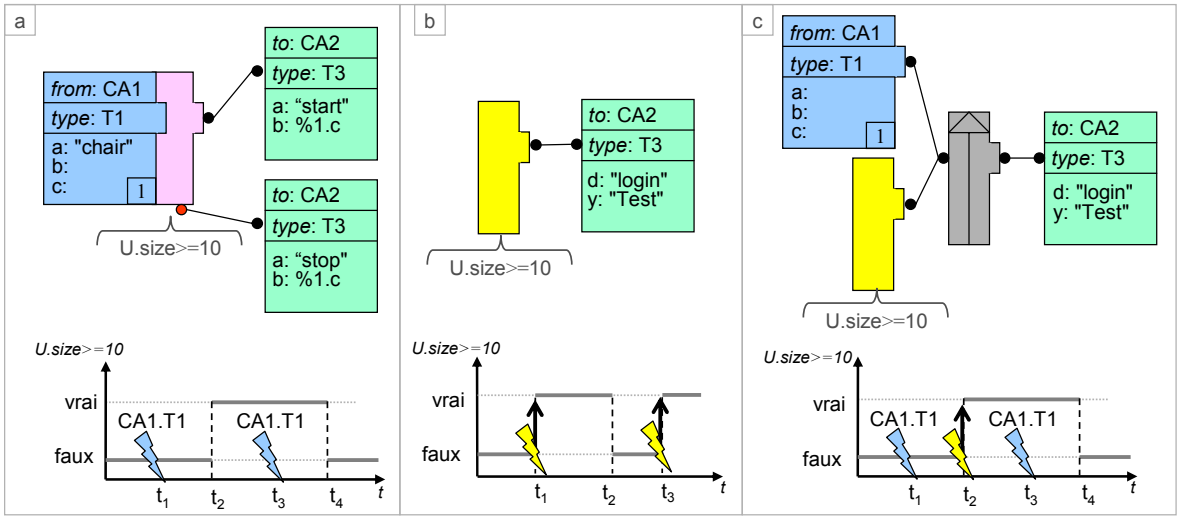

Figure IV.10 Exemples de règles portant sur des prédicats

Le *widget Predicate* contient un prédicat qui peut exprimer des contraintes :

(i) sur l'état de la *SuperSession* – le prédicat peut faire référence à des composantes de l'état global de la *SuperSession* ;

(ii) de temps – le temps peut être considéré comme une composante de l'état global de la *SuperSession* représentant la durée de la session ;

(iii) des paramètres des *Event*s définis dans la règle – le prédicat peut alors permettre de contraindre les paramètres des *Events* (nous verrons ultérieurement les contraintes d'utilisation de ces paramètres).

Un autre *widget*, le *Trigger*, permet également de définir une condition de sensibilisation d'une règle. Contrairement au *Predicate*, ce *widget* est évalué continuellement, et à chaque fois que la condition devient *vraie*, le *widget* devient sensibilisé. Remarquons que le *Trigger* peut définir une condition portant uniquement sur des composantes de l'état global de la *SuperSession* (y compris sa durée). Comme dans l'exemple "b" de la Figure IV.10, selon l'axe temporel illustré au dessous de la règle, la condition définie par le *Trigger* devient *vraie* en t_1 et en t_3, sensibilisant le *widget* et sensibilisant par voie de conséquence la règle à ces deux instants. Le *Trigger* a ainsi la sémantique d'un événement qui peut "notifier" des changements de l'état global de la *SuperSession*.

Dans l'Annexe B (section B.3.2), nous présentons les détails de la syntaxe définie pour la spécification des conditions pour les *widgets Predicate* et *Trigger*.

Les *widgets Earliest* et *Latest* permettent de composer différentes notifications d'événement lors de la spécification d'une règle politique.

Quand des *widgets* sont regroupés par un *Earliest*, le *Earliest* est sensibilisé dès qu'un des *widgets* spécifiés devient sensibilisé. Ainsi, dans l'exemple "c" de la Figure IV.10, quand la condition du *Trigger* devient *vraie* en t_2, le *Earliest* est sensibilisé et par conséquent la règle est sensibilisée ; cette règle est également sensibilisée en t_1 et t_3, suite aux autres notifications d'événement.

Quand des *widgets* sont regroupés par un *Latest*, le *Latest* est sensibilisé dès que tous les *widgets* spécifiés sont sensibilisés. Ainsi dans une règle simple composée juste par un *Latest* regroupant des *Event*s (comme dans la Figure IV.11), la règle politique est sensibilisée dès que tous les événements spécifiés sont notifiés.

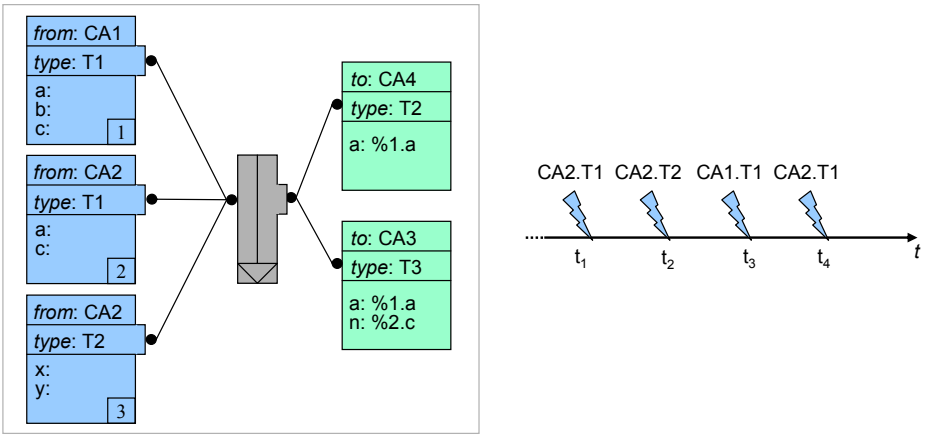

Figure IV.11 Analyse d'un Latest

Une particularité du *Latest* concerne ce que nous appelons le "temps d'attente", ou t_w. Chaque *Latest* a un t_w associé (paramétrable) qui définit l'écart maximal accepté entre les instants de la première et de la dernière notification d'événement spécifiées par le

Latest. Par exemple, nous avons illustré dans la Figure IV.11 un *Latest* qui regroupe trois *Event*s. Supposons que nous ayons la séquence de notifications d'événements représentée sur l'axe temporel situé à droite de la figure :

- Hypothèse 1 : $t_3 \leq (t_1 + t_w)$
  Dès que l'application "CA2" notifie l'événement "T1" en t_1, nous attendons jusqu'à $t_1 + t_w$ que les autres événements soient notifiés. En t_3, les trois événements spécifiés ont été notifiés et la règle devient donc sensibilisée. En t_4, nous recommençons et attendons jusqu'à $t_4 + t_w$ que les autres événements soient également notifiés.
- Hypothèse 2 : $t_3 > (t1 + t_w)$
  Dès que l'application "CA2" notifie l'événement "T1" en t_1, nous attendons jusqu'à $t_1 + t_w$ pour que les autres événements soient notifiés. En $t_1 + t_w$, puisqu'il manque toujours la notification de l'événement "T1" en provenance de "CA1", l'événement notifié en t_1 est rejeté, et nous considérons t_2 comme constituant le nouveau point de référence. Nous attendons donc jusqu'à $t_2 + t_w$ pour que les autres événements soient notifiés. Supposant $t_4 \leq t_2 + t_w$, en t_4 les trois événements spécifiés ont été notifiés et la règle est donc sensibilisée.

Dans le cas où nous avons un *Predicate* couplé à un *Earliest*, comme cela est illustré dans la Figure IV.12, dès qu'un des événements composés par le *Earliest* est notifié, ce dernier est sensibilisé et le prédicat est évalué. Selon la figure ci-dessous, des notifications d'événement se produisent en t_1, t_3 et t_4, sensibilisant ainsi le *Earliest*. A chacun de ces instants, la condition est évaluée : en t_3 (ainsi qu'en t_4) la condition est vraie, la règle est sensibilisée et donc tirée. Remarquons que *Earliest* définit un comportement non déterministe ; nous ne pouvons donc pas savoir quel *Event* parmi ceux qu'il regroupe sera notifié en premier, et par voie de conséquence, dans cette situation, le *Predicate* ne peut pas faire référence aux paramètres associés aux *Event*s. Pour la même raison, les *widgets Action*s ne peuvent pas non plus référencer les paramètres des *Event*s.

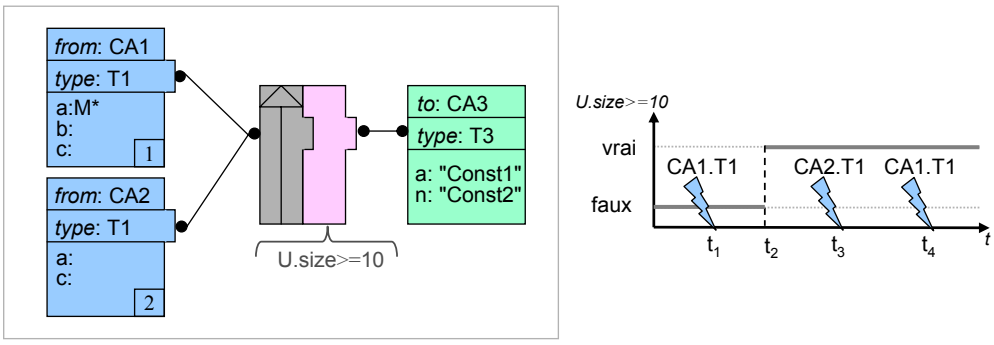

Figure IV.12 Exemple d'association d'un prédicat à un Earliest

Un *Predicate* peut également être couplé à un *Latest*. Par contre, nous avons décidé de traiter cette composition différemment des compositions précédentes dans le but de donner une sémantique plus riche aux règles. Ainsi, lorsque nous avons un *Latest* couplé à un *Predicate,* nous considérons ces deux *widgets* comme constituant un seul *widget* composé, que nous appelons *Latest+Predicate*.

Pour que le *Latest+Predicate* soit sensibilisé, il faut que deux contraintes soient satisfaites. D'abord, tout les *widgets* regroupés par le *Latest* doivent être sensibilisés. Ensuite, il faut que la condition imposée par le *Predicate* soit également satisfaite. Prenons par exemple le cadre "a" de la Figure IV.13. Dans ce cas, après les notifications des deux

événements attendus en t_1 et en t_2[1], le *Latest* devient sensibilisé et le prédicat est évalué. Si le prédicat était vérifié, le *Latest+Predicate* serait sensibilisé. Par contre, comme nous l'avons illustré dans l'axe temporel au dessous de la figure, les deux notifications reçues en t_1 et en t_2 véhiculent des valeurs de paramètres qui entraînent une évaluation à faux du prédicat. Au lieu de rejeter ces notifications, nous attendons que de nouveaux événements soient notifiés jusqu'à ce que le temps t_w arrive à échéance pour chaque notification. Nous voyons ici tout l'intérêt de ce comportement : de cette façon, nous pouvons attendre la notification de nouveaux événements véhiculant d'autres valeurs de paramètres. Ainsi en t_3, lorsque nous avons une deuxième notification provenant de "CA2", le *Latest* redevient sensibilisé et le prédicat est évalué de nouveau. Mais puisque nous avons eu deux notifications en provenance de "CA2"[2], la condition est évaluée pour chaque couple de notifications :

(i) "CA1.T1(a=x)" et "CA2.T1(a=y)" ;

(ii) "CA1.T1(a=x) et "CA2.T1(a=x)".

Pour (ii), la condition est vraie et le *Latest+Predicate* est sensibilisé, sensibilisant également la règle. Remarquons que, puisque ce sont les notifications reçues en t_1 et t_3 qui ont occasionné la sensibilisation de la règle, la notification reçue en t_2, n'est pas rejetée. Elle reste au contraire dans la "mémoire" du *Latest* jusqu'à ce que t_w arrive à échéance. Par conséquent, comme illustré dans la figure, en t_4 la règle sera encore sensibilisée[3] à cause de la nouvelle notification en provenance de "CA1".

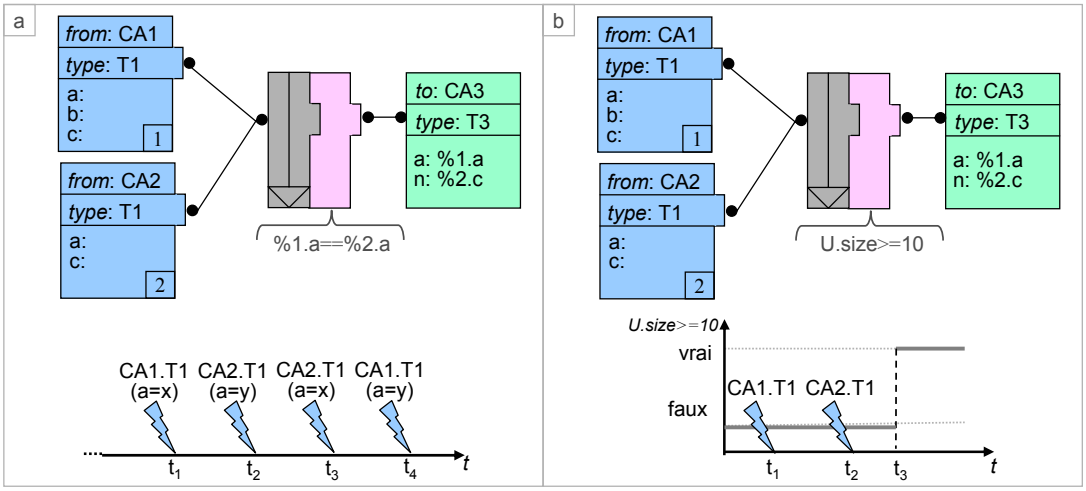

Figure IV.13 Exemple d'association de prédicats à des Latests

Une remarque à faire est qu'en définissant ce comportement pour le *Latest+Predicate* nous risquons de donner au *Predicate* la même sémantique qu'un *Trigger*. En effet, comme nous l'illustrons dans le cadre "b" de la Figure IV.13, après les deux notifications arrivées en t_1 et en t_2, puisque le prédicat porte sur une composante de l'état global de la *SuperSession*, il peut être évalué à vrai avant que de nouvelles notifications arrivent (suite à d'éventuels changements dans l'état global de la *SuperSession*), occasionnant ainsi la sensibilisation du *Latest+Predicate* en t_3. Afin d'éviter ce comportement, pour qu'un *Predicate* puisse être couplé à un *Latest*, le prédicat ne doit porter que sur des paramètres des événements regroupés par le *Latest* (et

[1] En supposant que $t_2 \leq t_1 + t_w$.
[2] Supposant $t_3 \leq t_1 + t_w$.
[3] Supposant $t_4 \leq t_2 + t_w$.

éventuellement des composantes *statiques* de l'état global[1]). Ainsi, l'exemple "b" de la Figure IV.13 n'est pas valide.

Dans la Figure IV.14 nous montrons comment des *Earliest*s et *Latest*s peuvent être combinés afin de définir des règles composées. Remarquons que seuls les paramètres de l'*Event* "3" sont référencés dans les *Actions*, vu que cet événement est le seul dont la notification est assurée lors de la sensibilisation de cette règle (il n'est pas possible de déterminer si c'est la notification d'un événement "CA1.T1" ou "CA1.T2" qui est à l'origine de la sensibilisation du *Earliest*).

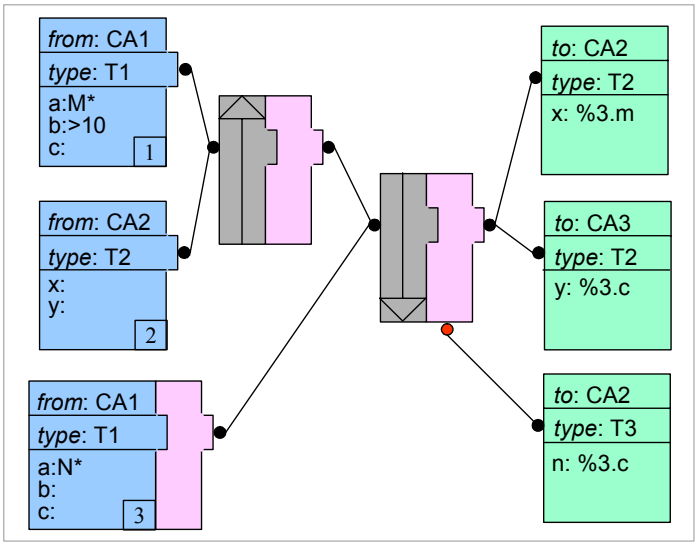

Figure IV.14 Exemple de règle composée

IV.4.2.3. Exemples de règles

Afin d'illustrer l'application d'une politique de collaboration dans une *SuperSession*, revenons sur le scénario de *E-learning* présenté dans la section IV.2.1.3.

Le premier comportement que nous pouvons exprimer à travers les règles concerne la connexion initiale des utilisateurs aux applications collaboratives. Dans le scénario de *E-learning*, nous considérons le monde virtuel comme point de départ de la *SuperSession*. Cela signifie que lorsqu'un utilisateur se connecte à la *SuperSession*, il ne doit être initialement connecté qu'au CVE. La Figure IV.15 illustre la règle implémentant ce comportement.

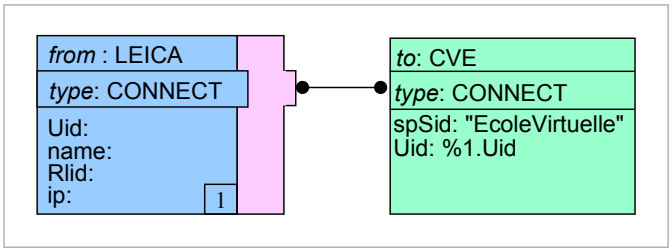

Figure IV.15 Connexion initiale d'un utilisateur au monde virtuel

[1] Les composantes statiques de l'état correspondent aux entités dites statiques de la *SuperSession*.

Dans ce scénario d'intégration nous avons prévu une session spécifique de tableau blanc partagé où le professeur et les moniteurs peuvent discuter ensemble, par exemple, au sujet des supports de cours. Pour se connecter au tableau blanc, nous avons indiqué que les utilisateurs doivent déplacer leur avatar respectif vers la salle des enseignants. Dans la Figure IV.16, nous illustrons la règle qui implémente ce comportement, ainsi que la déconnexion des utilisateurs lorsqu'ils quittent la salle des enseignants.

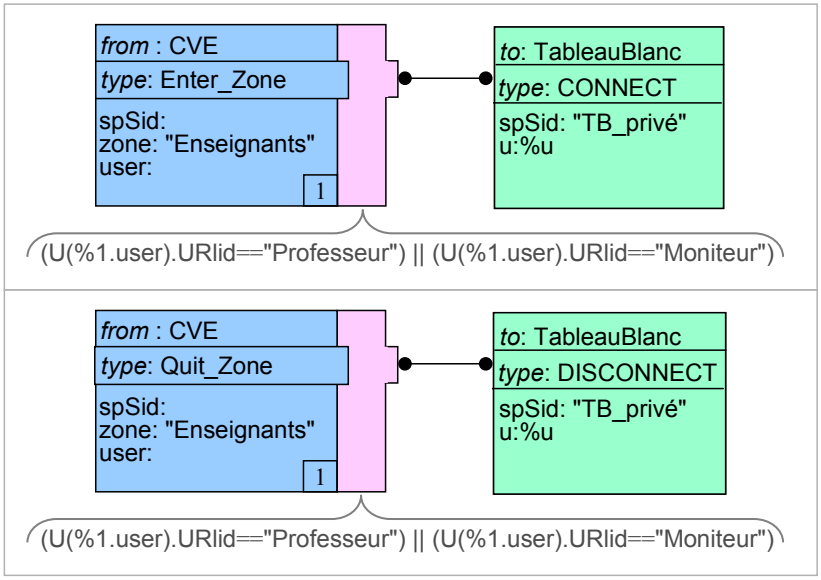

Figure IV.16 La connexion/déconnexion automatique de moniteurs et professeurs à la session de tableau blanc

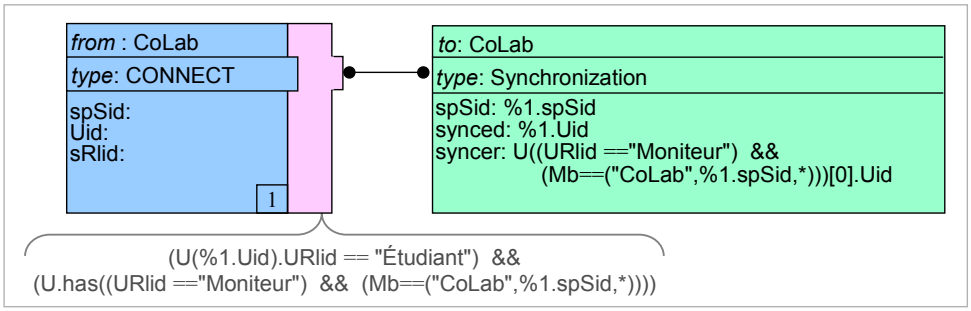

Figure IV.17 Une règle définissant la synchronisation automatique des utilisateurs dans CoLab

Un aspect intéressant des politiques de collaboration est qu'en plus de permettre le contrôle des interactions entre les applications intégrées à LEICA, elles permettent également de modifier certains comportements internes à ces applications. Regardons par exemple les cours virtuels dont la présentation des supports est assurée par *CoLab* et l'outil d'audioconférence. Dans *CoLab*, chaque fois qu'un utilisateur se connecte à une session de navigation coopérative, il démarre son activité de façon asynchrone (c'est-à-dire, il garde initialement l'autonomie de sa navigation). Nous voudrions maintenant changer ce comportement de façon à synchroniser automatiquement les utilisateurs de rôle "Étudiant" à un utilisateur de rôle "Moniteur" (bien évidement, s'il en existe un connecté à cette session spécifique). La Figure IV.17 illustre une règle permettant d'implémenter ce nouveau comportement. Dans le *Predicate* de cette règle nous testons (i) si l'événement notifié concerne un utilisateur de rôle général "Étudiant", et (ii) s'il existe un utilisateur de

rôle général "Moniteur" dans cette session spécifique de *CoLab*. Puis, dans le *widget Action*, il suffit de forcer la synchronisation de ces deux utilisateurs.

Nous pouvons également implémenter un comportement permettant l'attribution automatique du *floor* de l'outil d'audioconférence (c'est-à-dire, le droit de parole) à celui qui "guide" la navigation coopérative dans *CoLab*. Ce comportement représente une sorte de *floor coupling* (ou de couplage de *floor*) entre ces deux outils. Par contre, *CoLab* n'offre pas de notion explicite de floor de navigation. En fait, tous les utilisateurs qui sont asynchrones disposent chacun de leur propre *floor* de navigation. Comme nous l'avons décrit dans le paragraphe précédent, les seuls utilisateurs à garder leur autonomie dans les sessions spécifiques de *CoLab* sont les "Moniteurs". Par conséquent, la solution la plus simple pour implémenter ce comportement consiste à associer directement le *floor* de l'audioconférence à ceux qui sont associés à ce rôle. La règle définissant ce comportement est illustrée par l'exemple "a" de la Figure IV.18. Nous pourrions encore étendre ce scénario, de façon à permettre que lorsqu'un "Moniteur" passe temporairement le contrôle de la navigation Web à un autre utilisateur, on passe automatiquement le *floor* de l'audioconférence à ce même utilisateur. Dans *CoLab*, ce "passage" de contrôle de la navigation se réalise quand le "Moniteur" vient à suivre un autre utilisateur (par conséquent, tous ceux qui suivent déjà le "Moniteur" sont indirectement synchronisés avec cet utilisateur). Pour implémenter le passage automatique du *floor* dans l'outil d'audioconférence, il suffit d'associer une requête d'exécution d'action à cet événement, tel que nous le montrons dans l'exemple "b" de la Figure IV.18.

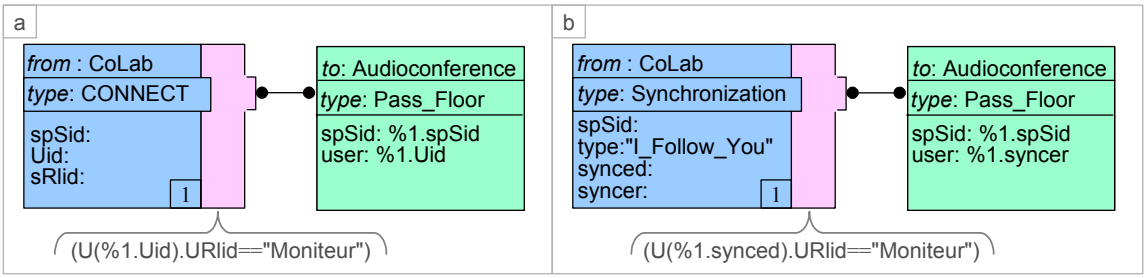

Figure IV.18 Des règles implémentant le couplage de floor entre CoLab et l'audioconférence

IV.4.2.4. Formalisation de la sémantique des règles politiques

Dans cette section nous proposons une démarche pour donner une sémantique formelle aux règles de collaboration. Remarquons que notre objectif est uniquement de définir la sémantique expliquant l'exécution des règles, et non pas de chercher à faire des analyses formelles ou des validations qualitatives/quantitatives du système d'exécution des règles de LEICA. Dans cette démarche nous avons choisi de nous appuyer sur les réseaux de Petri et leurs extensions [Diaz-01] [Diaz-03].

L'un des intérêts d'utiliser des réseaux de Petri (RdP) pour définir une telle sémantique réside dans le fait que, de façon intuitive, leur représentation graphique se rapproche de celle des règles de collaboration spécifiées pour LEICA. En outre, les RdPs constituent un modèle théorique simple qui permet de décrire et d'analyser des systèmes à événements discrets. Un RdP se caractérise par une évolution asynchrone dans laquelle les transitions sont franchies les unes après les autres, entraînant la dynamique des jetons dans le réseau. Cette caractéristique est très intéressante pour modéliser les aspects

événementiels associés à l'exécution des règles spécifiées dans la politique de collaboration d'une *SuperSession*.

Les RdPs de base ne sont cependant pas suffisamment expressifs pour représenter, de manière simple, toute la complexité associée aux règles politiques. Ceci s'explique notamment par le fait que les jetons dans les RdPs de base n'ont pas d'identité. Pour surmonter ce problème, des réseaux appelés "réseaux de Petri de haut niveau" permettant de considérer des jetons portant une identité, ou même des valeurs distinctes, ont été proposés. Parmi ces RdPs, nous trouvons les réseaux de Petri Colorés [Jensen-92], les réseaux de Petri Prédicats/Transitions [Genrich-81] et les réseaux de Petri à Objets [Gudwin-98].

Dans un réseau de Petri de haut niveau, le réseau modélise la structure de contrôle du système tandis que les jetons modélisent les données, ces derniers pouvant représenter des structures de données de différents niveaux de complexité tels que des entiers, des chaînes de caractères, des listes et des tuples. Un autre aspect des réseaux de Petri de haut niveau est que les arcs reliant les places aux transitions sont étiquetés par une fonction qui permet de spécifier la nature des jetons retirés ou mis dans une place lors du franchissement d'une transition. De plus, il est possible d'associer aux transitions des conditions sous la forme d'expressions booléennes ou de prédicats.

Puisque les règles politiques peuvent également prendre en compte des aspects temporels (notamment à cause de la définition du t_w dans le *widget Latest*), nous devons considérer une extension supplémentaire des RdPs, à savoir les RdP temporels, ou RdPT [Merlin-76] [Berthomieu-91]. Ces réseaux de Petri permettent d'exprimer des contraintes temporelles en associant des intervalles temporels aux transitions. A chaque transition est ainsi associé un intervalle temporel $[t_{min}, t_{max}]$ où t_{min} indique la durée minimale pendant laquelle une transition doit être continuellement sensibilisée avant d'être franchie, et t_{max} la durée maximale à l'échéance de laquelle la transition doit être nécessairement franchie si elle est restée continuellement sensibilisée.

Finalement, pour compléter les fondements théoriques nécessaires à la définition de la sémantique des règles politiques, nous sommes allés chercher dans les travaux de Keller [Keller-76] la possibilité de définir des variables d'état globales dont les valeurs sont toujours disponibles, indépendamment de l'état du réseau de Petri. Nous voyons dans notre cas l'importance de ce mécanisme étant donné que les règles peuvent faire référence à des prédicats portant sur des composantes de l'état global d'une *SuperSession*.

Nous avons choisi de modéliser une règle de collaboration par un RdPT de haut niveau (par simplicité, nous l'appellerons, un réseau de Petri ou simplement un RdP) :

- le réseau représente la structure de contrôle de la règle définissant comment des notifications d'événement conduisent à l'exécution des actions ;
- les jetons représentent les notifications d'événements et véhiculent des données relatives au type d'événement et aux valeurs de ses paramètres ;
- des prédicats, portant sur des variables locales à une transition (valeurs véhiculées dans les jetons) ou des variables globales (composantes de l'état global de la *SuperSession*) peuvent être associés aux transitions.

Vue la nature compositionnelle des règles, nous avons adopté une approche similaire à celle proposé dans [Sadani-05] pour construire les réseaux de Petri. La démarche consiste à structurer le réseau de Petri en composants. Le réseau de Petri final est le résultat de la composition de ses composants. Ainsi, pour engendrer un réseau de Petri caractérisant la sémantique d'une règle, nous considérons chaque *policy widget* et

appliquons un schéma de traduction pour engendrer un composant encapsulant un RdP. Ensuite nous appliquons un schéma de composition qui consiste à fusionner des places spécifiques des composants. Un composant encapsulant un RdP définit, en plus des places simples, deux ensembles spéciaux de places : (i) $I=\{In_i\}$ qui définissent les interfaces d'entrée du composant ; et (ii) $O=\{Out_i\}$ qui définissent les interfaces de sortie du composant. Le schéma général de composition consiste alors à fusionner les interfaces d'entrée et de sortie des composants.

Dans la Figure IV.19 nous considérons le cas des *policy widgets Event*, *Trigger*, et *Action*. Dans les cas de *Event* et de *Trigger*, les transitions engendrent des jetons de type *Evt*, qui est un type composé de deux champs :

```
type Evt
        type : String
        p : String []
```

Le premier champ caractérise la source et le type de l'événement notifié. Le second champ comprend la liste de paramètres véhiculés dans la notification d'événement. Rappelons que les noms et types des paramètres des événements notifiés varient en fonction du type de l'événement. Nous avons choisi une représentation générale où toutes les notifications d'événement sont représentées par le même type de jeton *Evt*, et les valeurs des paramètres sont véhiculées dans un vecteur de valeurs[1] (l'ordre des valeurs correspondant à l'ordre dans laquelle chaque paramètre apparaît dans la spécification de l'événement[2]). Les valeurs des paramètres, qui sont à l'origine numériques (*i.e.* entiers, flottants, *etc.*) sont automatiquement converties en une représentation textuelle.

La structure du composant RdP associé à un *widget Event* est constituée d'une transition et d'un arc (annoté par *ei* pour indiquer le type de jeton engendré) liant la transition à la place *Out*. Par cette transition sans place d'entrée, nous exprimons le fait que la notification d'un événement arrive de manière "inconditionnelle". L'intervalle *[0,0]* associé indique que la transition est franchie dès que l'événement attendu est notifié. Puisque les notifications d'événement sont la conséquence d'une communication entre applications collaboratives, nous empruntons pour noter une notification d'événement une notation utilisée dans la communication interprocessus [Bolognesi-87]. Ainsi, *P?v:T[Pr]* indique la réception d'une valeur de sorte *T* sur la porte *P*, un prédicat de sélection *Pr* pouvant être spécifié pour imposer des contraintes sur la valeur reçue ; cette valeur reçue est affectée à la variable *v*.

Ainsi, une notification d'événement est notée de la manière suivante :

```
CA ?ei.type:String[ei.type='T']
   ?ei.p[0]:String[Pr(P0)] ...
   ?ei.p[n]:String[Pr(Pn)]
```

où:

- *CA* représente la porte d'origine (c'est-à-dire, l'application collaborative source de l'événement) ;
- *ei.type* reçoit la valeur *T* ;

[1] Originalement chaque paramètre peut être défini en tant qu'un vecteur de valeurs. Par simplicité, nous allons considérer dans la formalisation les paramètres ne véhiculant que des valeurs simples.
[2] Par simplicité nous omettons le paramètre "spSid" défini par défaut pour tout événement.

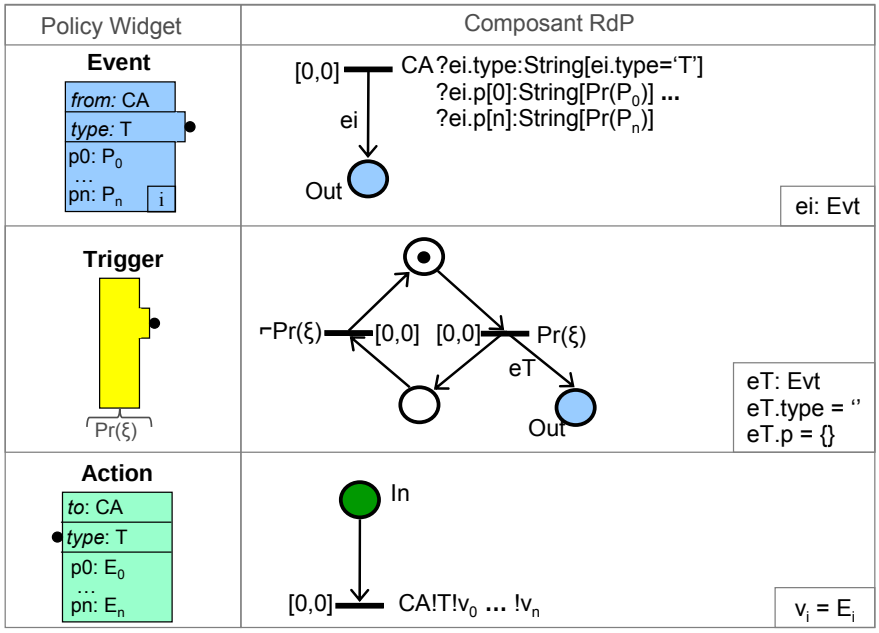

Figure IV.19 Composants RdP associés aux widgets Event, Trigger et Action

- *ei.p[i]* reçoit une valeur (correspondant au paramètre *pi* du *widget*) et filtré par le prédicat $Pr(P_i)$ ($Pr(P_i)$ est un prédicat au niveau de la sémantique qui implémente le *matching pattern* P_i).

La structure du composant RdP associé à un *widget Trigger* est un peu plus élaborée que celle du *widget Event*. Ce composant RdP présente une boucle entre les deux transitions annotées par $\neg Pr(\xi)$ et $Pr(\xi)$ respectivement, où ξ est le vecteur des variables représentant les composantes de l'état global de la *SuperSession*. Initialement, seule la place d'entrée de la deuxième transition (celle de droite) est marquée. Cette transition représente une transition spontanée qui est immédiatement tirée (voir la présence de l'intervalle *[0,0]*) lorsque le prédicat $Pr(\xi)$ devient vrai. Suite à ce tir, deux jetons sont engendrés : un jeton *eT*, mis dans la place *Out*, et un jeton simple (*i.e.* un jeton associé à aucun type de donnée particulier), mis dans la place d'entrée de la transition annotée par $\neg Pr(\xi)$. Il faut donc que cette transition (celle de gauche) soit tirée, c'est-à-dire, que $Pr(\xi)$ devienne faux, pour que le RdP revienne au même état initial et que d'autres jetons puisse être engendrés dans la place *Out*.

Remarquons que le jeton *eT* placé dans *Out* ne véhicule pas de valeur particulière. En fait, nous n'avons pas choisi d'engendrer des jetons simples parce que les places d'un RdP de haut niveau doivent toujours avoir des jetons d'une même sorte. Nous verrons ultérieurement, que dans certaines situations (*par exemple* à cause de la présence d'un *Latest* dans une règle), des jetons engendrés par des *Triggers* et des *Events* peuvent être regroupés dans des places communes.

Le dernier *widget* illustré dans la Figure IV.19 est le *widget Action*. La structure du composant RdP associé est composée d'une place d'entrée et d'une transition. L'intervalle *[0,0]* associé à la transition indique que la transition est franchie dès qu'elle est sensibilisée. Puisque l'action représente l'envoi d'un message à une application collaborative, nous empruntons de nouveau la notation utilisée dans une communication interprocessus. Ainsi, *P!E* caractérise l'émission d'une valeur décrite par l'expression *E* sur la porte *P*. Nous associons ainsi à la transition la notation suivante :

$$CA!T!v_0 \ldots !v_n$$

où:

- *CA* représente la porte sur laquelle les valeurs sont rendues disponibles (c'est-à-dire, l'application collaborative destinatrice) ;
- *T* est la première valeur, qui correspond au type d'action dont l'exécution est sollicitée ;
- $!v_0$ ... $!v_n$ sont les valeurs émises qui correspondent aux paramètres définis pour cette action (dans l'ordre où les paramètres sont définis, c'est-à-dire, $p_i = v_i$).

Chaque valeur v_i est décrite par une expression de valeur E_i spécifiée pour le paramètre *pi* du *widget Action*, E_i étant en général une fonction $f(\xi,\varepsilon)$. Nous définissons ε comme l'ensemble des jetons retirés de la place *In*. Cet ensemble n'aura d'éléments qu'après la composition des RdPs correspondant à l'ensemble des *policy widgets* d'une règle. En fait, pour permettre la propagation des jetons à travers le RdP final, en règle générale, lorsque nous fusionnons une place *Out* d'un composant à une place *In* d'un autre composant RdP, chaque transition en sortie de cette place *In* est étiquetée avec la même étiquette (*i.e.* les mêmes types de jetons[1]) que celle de la transition en entrée de la place *Out*[2]. Dans le cas où le composant n'a qu'une place *In*, nous étiquetons tous les arcs du RdP contenant cette place *In* avec la même étiquette.

Dans la Figure IV.20, nous reprenons la règle illustrée dans la Figure IV.9 où nous avons une association simple d'un *widget Event* et d'un *widget Action*. Les composants *C1* et *C2* sont identifiés (chacun correspondant à un *policy widget*) et ensuite composés, produisant le RdP à droite de la figure.

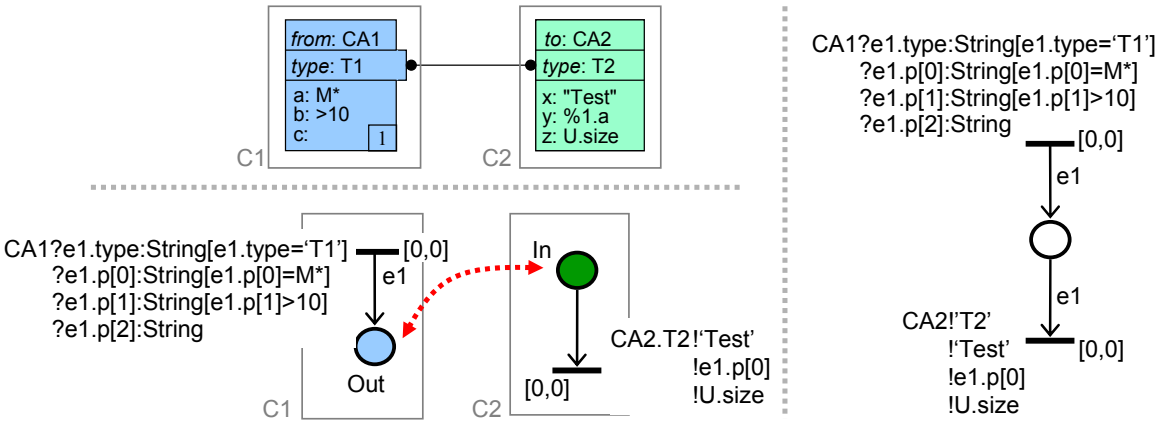

Figure IV.20 Composition simple d'un widget Event et d'un widget Action

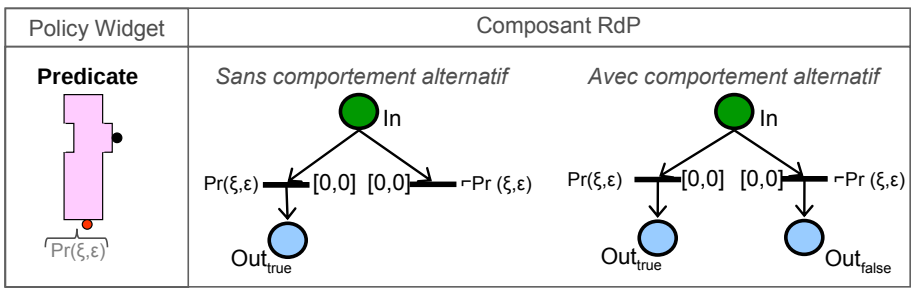

Figure IV.21 Les composants RdPs associés au widget Predicate

[1] Cela peut représenter un seul jeton ou une somme formelle de jetons.

[2] Remarquons que chaque place *Out* des composants RdPs, à l'exception de celle associée au *Earliest*, est associée à une seule transition (nous verrons plus tard le cas spécifique du *Earliest*).

La Figure IV.21 illustre les composants RdPs associés au *widget Predicate*. Chaque RdP est formé par une place *In* liée à deux transitions conditionnées par les prédicats $Pr(\xi,\varepsilon)$ et $\neg Pr(\xi,\varepsilon)$ respectivement. Dans le cas où le *Predicate* n'a pas de comportement alternatif spécifié, seule la transition annotée avec $Pr(\xi,\varepsilon)$ est liée à la place Out_{true}. Si le comportement alternatif du *Predicate* est spécifié, la transition annotée avec $\neg Pr(\xi,\varepsilon)$ est également associée à une interface de sortie (Out_{false}).

Dans la Figure IV.22, nous reprenons une règle initialement illustrée dans le cadre "a" de la Figure IV.10. Nous schématisons la composition du RdP final (à droite de la figure) en composant les RdPs correspondant à chaque *policy widget* utilisé pour construire la règle.

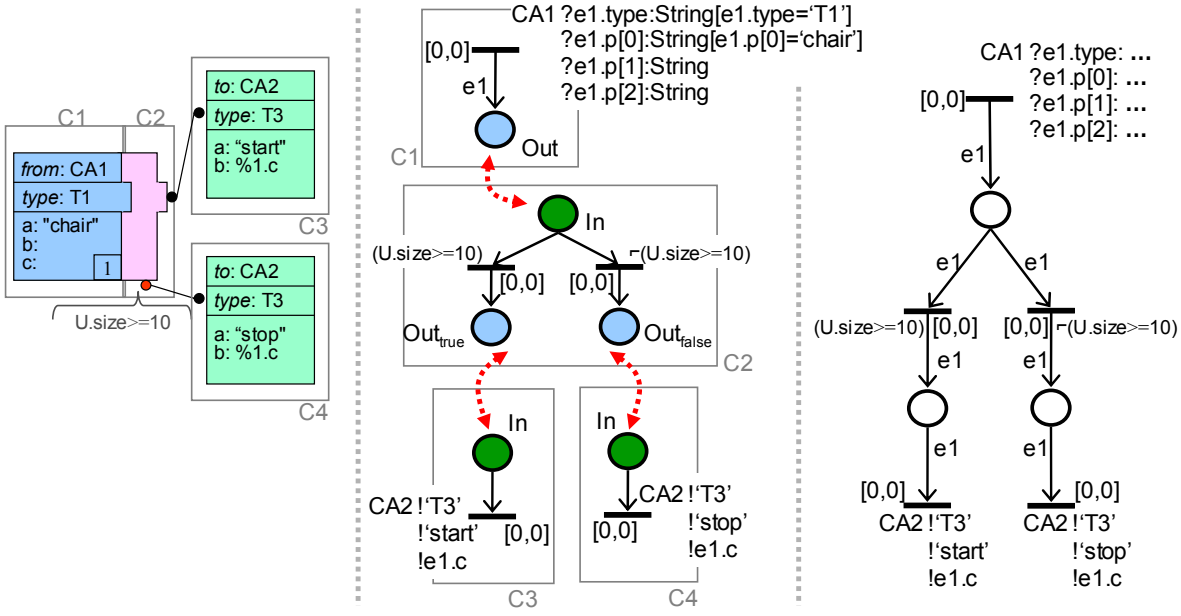

Figure IV.22 Composition d'un widget Event et d'un widget Predicate

Nous présentons finalement dans la Figure IV.23 les composants RdP associés aux *widget*s *Earliest* et *Latest*. Remarquons que ces composants présentent plus qu'une place *In*. Ainsi, la propagation des jetons lors d'une composition est traitée de manière différente.

Dans le cas du *Latest*, nous étiquetons d'abord les deux arcs de sortie de chaque place In_i avec la même étiquette que celle de la transition en entrée de la place *Out* qui sera fusionnée à In_i. Considérons que l'étiquette associée à cette transition définit une somme formelle de jetons Σ_i. Nous étiquetons alors l'arc d'entrée de la place *Out* du *Latest* avec la somme formelle $\Sigma_0 + ... + \Sigma_n$.

Dans le cas du *Earliest*, nous étiquetons chaque transition en sortie de chaque place In_i avec la même étiquette que celle de la transition en entrée de la place *Out* qui sera fusionnée à In_i. Par contre, les transitions en entrée de la place *Out* de ce composant sont étiquetées avec un jeton *eT* (comme dans le cas du *Trigger*, *eT* est un jeton qui ne véhicule pas de valeur particulière). Comme nous l'avons précisé auparavant, le *Earliest* présente un comportement non déterministe. Ainsi, la "mémoire" associée aux valeurs véhiculées par les jetons est effacée.

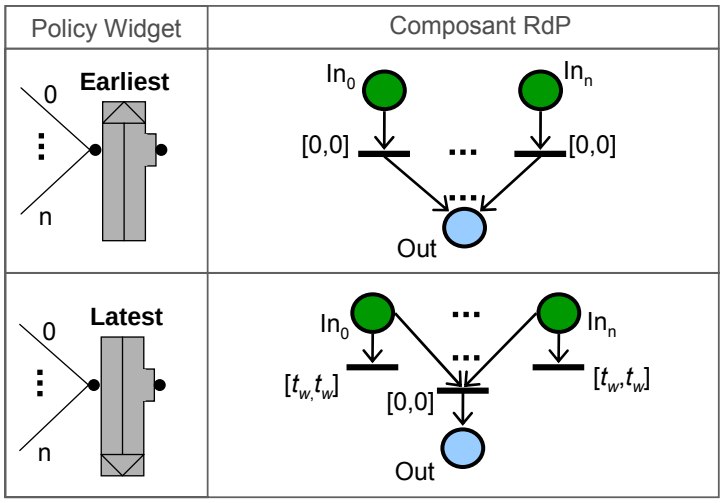

Figure IV.23 Les composants RdPs correspondant aux policy widgets Earliest et Latest

Rappelons que le composant RdP associé à un *Earliest* est le seul dont la place *Out* peut être associée à plus d'une transition. Nous avons ainsi une règle spécifique de propagation de jetons lorsque nous fusionnons une place *Out* d'un composant RdP associé à un *Earliest* à une place *In* d'un autre composant RdP, qui est la suivante : nous étiquetons toutes les transitons en sortie de la place *In* de l'autre composant avec *eT*. Ceci est illustré dans la Figure IV.24. Lorsque la place *Out* de "C3" est fusionnée à la place *In* de "C4", les deux transitions en sortie de cette place *In* sont étiquetées par *eT*.

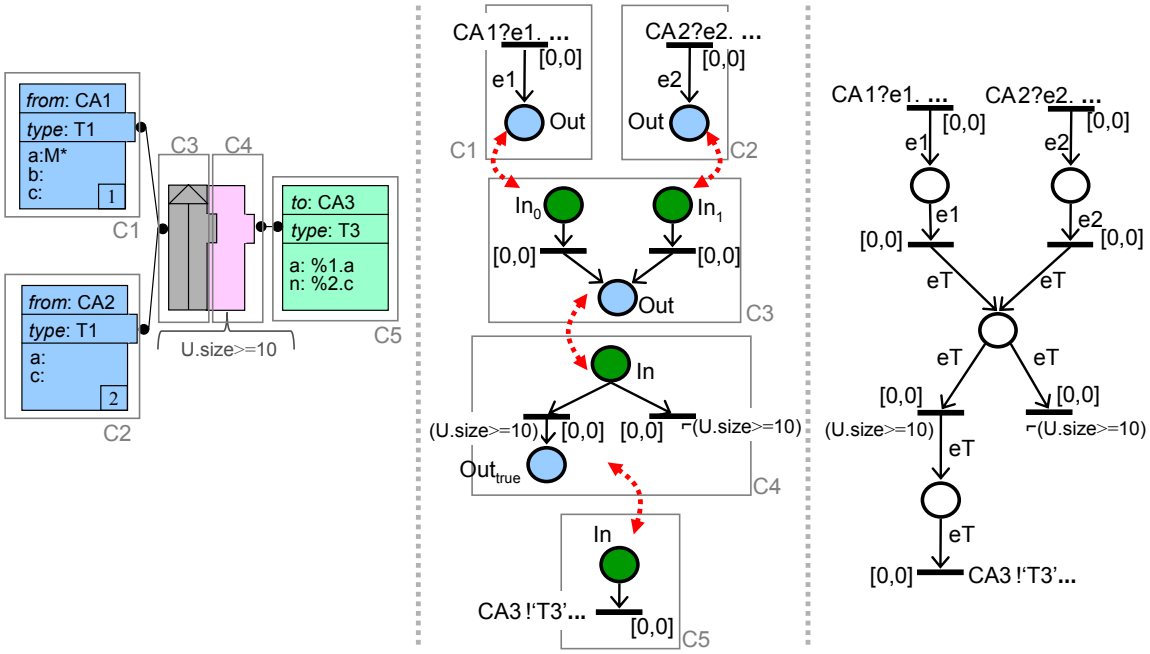

Figure IV.24 Exemple de composition en utilisant un Earliest et un Predicate

Dans la Figure IV.24 nous schématisons la composition d'un RdP correspondant à une règle comprenant un *Earliest* associé à un *Predicate*. Le RdP spécifie que lorsque nous avons une notification "CA1.T1" ou "CA2.T1", un jeton *e1* ou un jeton *e2* est engendré par l'une des deux transitions "source". Chaque jeton engendré perd alors son identité en se transformant en un jeton *eT*.

Par contre, comme nous l'avons expliqué dans la section IV.4.2.2, lorsqu'un *Latest* est couplé à un *Predicate*, nous considérons et traitons ces deux *widgets* comme un seul *widget* (*Latest+Predicate*). Ainsi, pour que le *Latest+Predicate* soit sensibilisé, il faut que (i) tous les *widgets* regroupés par le *Latest* soient sensibilisés[1] et que (ii) la condition imposée par le *Predicate* soit satisfaite. Le *widget Latest+Predicate* est sensibilisé dès que les deux conditions (i) et (ii) sont satisfaites en même temps.

Pour exprimer ce comportement, comme cela est illustré dans la Figure IV.25, nous représentons le *Latest* associé à un *Predicate*[2] sous la forme d'un seul composant RdP (au lieu de réaliser une composition des RdPs correspondant à chacun de ces *widgets*). Remarquons que les jetons sont retirés des places In_i en deux situations : (i) lorsque le *Latest* est sensibilisé et que le prédicat est vrai, et (ii) si nous avons un comportement alternatif spécifié dans le *Predicate*, lorsque le *Latest* est sensibilisé et que le prédicat est faux. Par contre, dans le cas où il n'y a pas de comportement alternatif spécifié, lorsque le *Latest* est sensibilisé et que le prédicat est faux, nous ne devons pas retirer les jetons, mais au contraire, nous devons attendre que d'autres jetons soient engendrés dans les places In_i (tout en respectant la contrainte temporelle t_w). Dans la Figure IV.26 nous illustrons un exemple de cette association où nous reprenons la règle initialement illustrée dans le cadre "a" de la Figure IV.13 et présentons le RdP composé exprimant la sémantique d'exécution de cette règle.

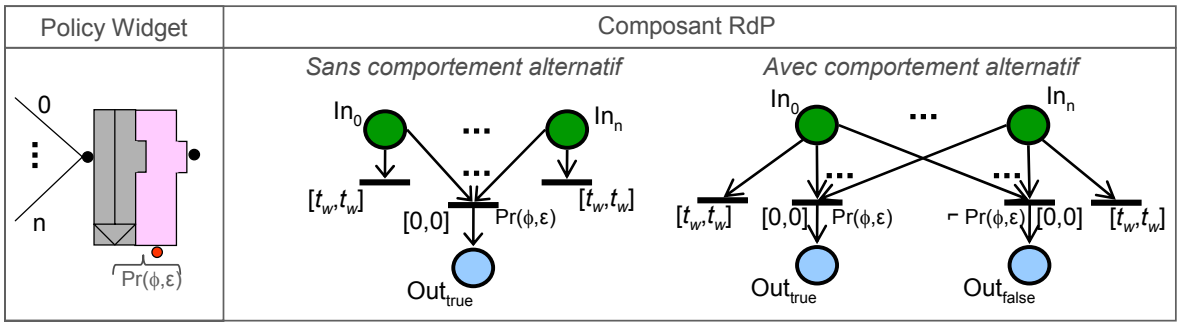

Figure IV.25 Le composant RdP associé à un Latest+Predicate

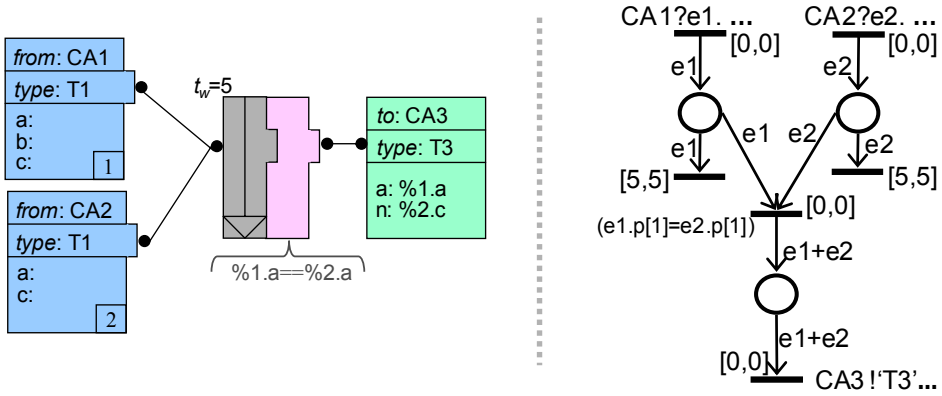

Figure IV.26 Exemple de règle utilisant un Latest+Predicate

[1] Par exemple, que toutes les notifications d'événement attendues arrivent.

[2] Le prédicat ne peut porter que sur ϕ et ε, où ϕ est le vecteur des variables représentant les composantes statiques de l'état global de la SuperSession.

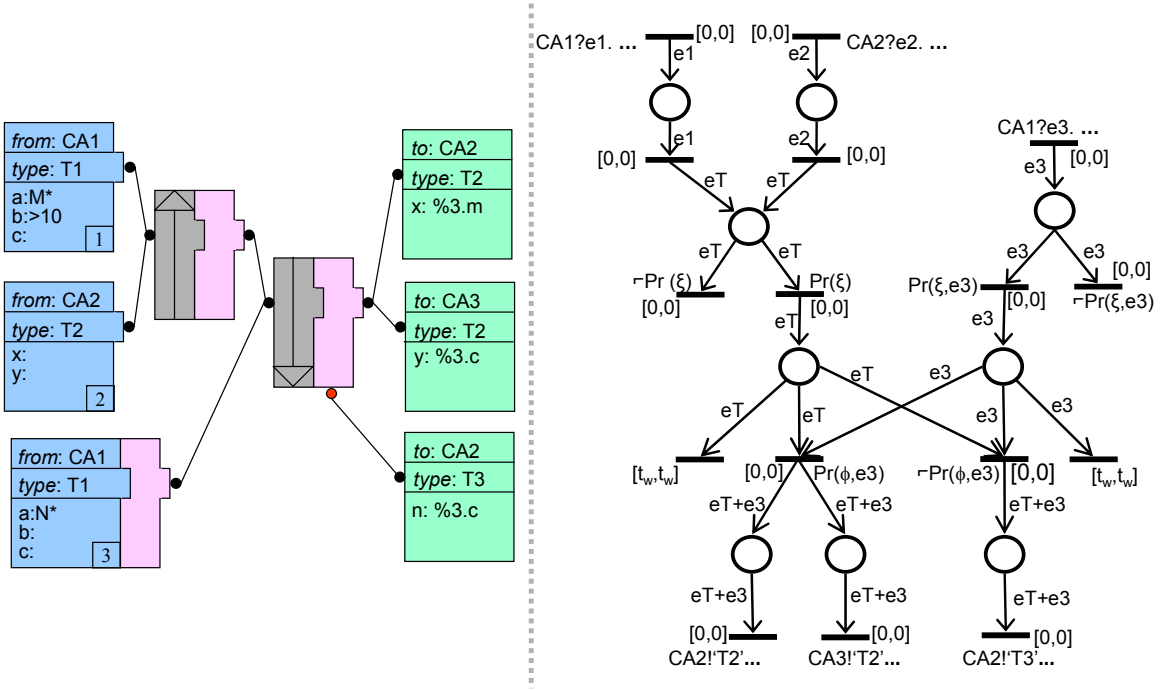

Figure IV.27 Un RdP correspondant à une règle plus complexe

Dans la Figure IV.27 nous présentons le RdP associé à une règle plus complexe (qui a été présentée auparavant dans la Figure IV.14). Remarquons que lorsque plus d'un *widget Action est* connecté à un même point de connexion, nous devons répliquer, avant la composition, les places *Out* (une par *Action*) du *widget* lié à ces *Actions*. Dans cet exemple, nous l'illustrons pour le *widget Predicate+Latest*.

IV.5. Conclusion

Dans ce chapitre nous avons présenté l'approche générale d'intégration proposée par LEICA. Nous avons expliqué cette approche en abordant les différentes étapes nécessaires à l'exécution d'une session collaborative globale (appelée *SuperSession*) utilisant l'environnement intégré. Ces étapes comprennent : (i) l'intégration de nouvelles applications à l'environnement, (ii) la définition d'une *SuperSession*, ainsi que (iii) l'exécution de cette *SuperSession*. Nous avons également présenté quelques principes architecturaux adoptés par LEICA, tel que le choix des standards de services Web pour réaliser les étapes initiales de l'approche d'intégration. Concernant l'exécution de la *SuperSession* elle-même, un service de notification d'événements est utilisé à la place des interfaces de services Web. Nous écartons ainsi les contraintes de performances liées à cette technologie, tout en gardant l'aspect faiblement couplé proposé par LEICA.

Après avoir donné une idée générale de l'environnement, nous avons présenté en détail le concept de *SuperSession*. La *SuperSession* représente l'abstraction de base comprenant tous les éléments d'une activité collaborative intégrée. Pour bien identifier ces différents éléments nous avons spécifié un modèle de session. S'appuyant sur ce modèle, nous avons pu définir les différentes composantes de l'état global d'une *SuperSession*. Ce modèle nous permet également d'inférer une taxonomie bien définie de ces composants.

Nous avons ensuite détaillé la configuration d'une *SuperSession*, qui se divise en deux partie: (i) la configuration des informations de gestion et la sélection et configuration

des applications pour le support de la *SuperSession*, et (ii) la spécification des politiques de collaboration (définition des règles politiques pour le contrôle des interactions entre les applications). Nous avons précisé la spécification des règles qui constituent un concept clé de notre système, permettant de définir des comportements automatiques pour la *SuperSession*. Nous avons également présenté la méthode de formalisation utilisée pour définir la sémantique des règles.

En s'appuyant sur l'approche général d'intégration de LEICA nous allons aborder dans les chapitres suivants l'architecture puis l'implémentation et le déploiement de l'environnement d'intégration proposé.

Chapitre V

L'architecture de LEICA

V.1. Introduction

Dans le chapitre précédent nous avons présenté l'approche générale d'intégration suivie par LEICA, ainsi que l'abstraction de base, dénommée la *SuperSession*, qui permet à LEICA de gérer une session collaborative en s'appuyant sur différentes applications intégrées à l'environnement. Ce chapitre, quant à lui, est consacré à la description de l'architecture de LEICA qui permet de mettre effectivement en œuvre l'approche générale d'intégration proposée. Notre objectif est donc de spécifier une architecture permettant notamment d'exécuter de manière répartie les *SuperSessions*.

Dans une première partie, nous présentons une vue globale de l'environnement d'intégration en décrivant les principaux modules de l'architecture ainsi que leurs rôles respectifs lors de l'exécution de l'environnement. Dans une deuxième partie, nous introduisons la méthode de modélisation utilisée lors de la conception de l'architecture de LEICA ; cette méthode s'appuie sur le profil UML/SDL [UML] [SDL] [Björkander-03] supporté par l'outil *Tau G2* de *Telelogic* [*Telelogic*].

V.2. Architecture : de l'intégration des applications à l'exécution d'une *SuperSession*

Dans le cadre général d'intégration de LEICA, présenté dans le chapitre précédent, nous avons introduit les trois activités principales nécessaires à l'environnement pour mettre en œuvre des *SuperSessions* :

(i) l'intégration d'applications à l'environnement,

(ii) la création de nouvelles *SuperSessions*, et

(iii) l'exécution de *SuperSessions*.

Pour illustrer l'architecture globale du système, et préciser le rôle des différents modules et la façon dont ils doivent interagir, nous allons détailler la mise en œuvre de chacune de ces activités.

V.2.1. Intégration d'une application collaborative à LEICA

Nous considérons trois différents types d'architecture pour les applications collaboratives à intégrer dans l'environnement LEICA : (i) des applications client/serveur, (ii) des applications multiserveurs, et (iii) des applications P2P.

Selon le type d'architecture, des *Wrappers* spécifiques doivent être ajoutés à ces applications afin de pouvoir les intégrer à l'environnement. Ce processus d'intégration est constitué de trois étapes ; il est décrit dans les sous-sections suivantes.

V.2.1.1. Création dynamique d'un *Wrapper*

Dans le but de diminuer les efforts de développement lors de l'intégration d'applications à LEICA, nous avons défini un module particulier, appelé *API Factory*. Ce module, en s'appuyant sur des spécifications XML des APIs d'une application, engendre un *Wrapper* adapté à cette application.

Pour chaque application collaborative on doit créer deux fichiers. Le premier, appelé *Specific Data File* (ou "fichier de données spécifiques"), définit quelles sont les données nécessaires à la création de sessions spécifiques pour cette application. Autrement dit, dans ce fichier on indique la liste des paramètres spécifiques dont cette application a besoin pour créer une session collaborative conventionnelle. Par exemple, dans le cas d'une application de vidéoconférence multicast, on doit spécifier l'adresse IP multicast associée à la vidéoconférence. Pour chaque paramètre, on spécifie son nom et son type.

Le deuxième fichier, appelé *Attributes and API File* (ou "fichier d'attributs et d'API"), définit deux groupes d'information :

(i) Des attributs caractérisant l'application collaborative
- l'identificateur unique de l'application collaborative ;
- le type d'application collaborative[1] ;
- s'il s'agit ou pas d'une application supportant la notion de rôle ;
- son architecture (client/serveur, multiserveurs, P2P) ;
- le type d'application utilisateur, c'est-à-dire, si l'application cliente (dans le cas d'une application client/serveur ou multiserveurs) ou l'application paire (dans le cas d'une application P2P) est une application *stand-alone* (installée sur le poste utilisateur) ou *web-based*, ainsi que les informations nécessaires pour exécuter ces applications (par exemple, dans le cas où l'application utilisateur est une application Web, on spécifie son URL et les éventuels paramètres).

(ii) L'API événements/actions de l'application collaborative – en s'appuyant sur l'API de l'application, on spécifie
- la liste des événements que l'application est capable de notifier ;
- la liste de requêtes qu'elle est capable d'exécuter, y compris celles concernant les actions de l'API de gestion (dans la section IV.2.1, nous avons expliqué quelles sont les informations qui doivent être fournies dans la spécification de l'API événements/actions d'une application).

Lors de la création dynamique d'un *Wrapper*, l'*API Factory* adapte un module *Wrapper* général à l'information décrite dans les deux fichiers XML créés pour l'application. Deux types de *Wrapper* peuvent être engendrés : un *Server Wrapper* (pour les applications collaboratives client/serveur ou multiserveurs) et un *P2P Wrapper* (pour

[1] Puisque ce paramètre ne sert qu'à orienter les administrateurs des *SuperSessions* dans leur choix des applications collaboratives à utiliser en support d'une *SuperSession*, nous n'avons pas défini une liste statique de types d'applications. Néanmoins, nous suggérons que les catégories fonctionnelles présentées dans la section II.2.2.2 soient utilisées pour typer les applications intégrées.

les applications P2P). La Figure V.1 illustre ces deux types de *Wrapper* en détaillant leurs sous-modules. La différence entre ces deux types de *Wrapper* concerne le sous-module *WS Interface*, responsable de l'implémentation des interfaces de services Web, qui n'existe que pour les *Server Wrappers*. A sa place, le *P2P Wrapper* contient le sous-module *P2P Proxy Interface*, qui implémente l'interface de communication avec le *P2P Proxy*.

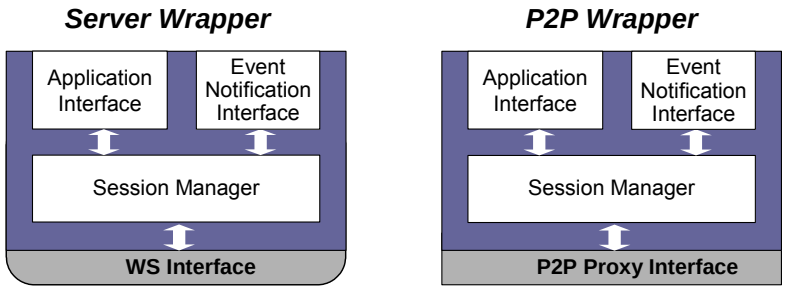

Figure V.1 Les deux Wrappers et leurs sous modules

L'adaptation d'un *Wrapper* consiste notamment à créer le sous-module *Application Interface*. Ce sous-module est le composant du *Wrapper* qui doit être couplé à l'application collaborative. Ainsi, par l'intermédiaire de ce sous-module, l'application collaborative pourra notifier au *Wrapper* "ce qui se passe" à l'intérieur de son contexte de collaboration, et elle pourra également recevoir des requêtes d'établissement de sessions spécifiques ainsi que des requêtes d'exécution d'actions. Cette communication entre l'application et LEICA se fait par le biais de méthodes définies dans l'*Application Interface*. Ces différentes méthodes, qui correspondent à l'API événements/actions et à l'API de gestion, sont schématisées dans la Figure V.2. Remarquons que le fait de définir des méthodes communes à toutes les applications (correspondant en fait à l'API de gestion[1]) représente cependant une contrainte vis-à-vis des applications pouvant être intégrées à LEICA. Autrement dit, nous ne pouvons intégrer que des applications qui supportent ces fonctionnalités dans leur API.

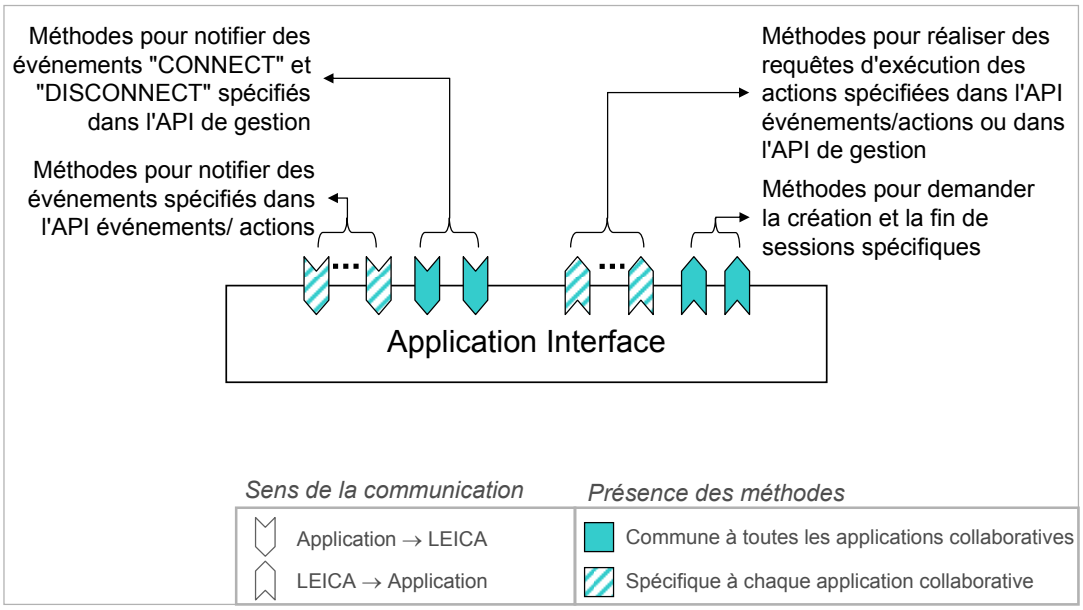

Figure V.2 Détail sur les méthodes définies dans le sous-module Application Interface

[1] En plus, nous avons les méthodes pour demander la création et la fin de sessions spécifiques.

Les sous-modules *Event Notification Interface* et *Session Manager* sont engendrés de manière statique pendant la création d'un *Wrapper* (*i.e.* ils sont donc identiques pour tous les *Wrappers*). Le premier implémente l'interface de communication entre le *Wrapper* et le système de notification d'événements, alors que le deuxième implémente toutes les fonctionnalités de base du *Wrapper*. Ce dernier a la responsabilité :

(i) de recevoir et traiter les données de configuration des *SuperSessions* et des sessions spécifiques respectives ;

(ii) d'envoyer les notifications d'événement (communiquées par l'application par l'*Application Interface*) aux autres applications collaboratives ;

(iii) de gérer les politiques de collaboration des *SuperSessions*, et de traiter les requêtes d'exécution des actions suite à l'exécution des règles politiques ;

(iv) de gérer l'état des *SuperSessions*.

V.2.1.2. Couplage du *Wrapper* à l'application

Une fois qu'on a créé le *Wrapper* spécifique à une application, on doit le coupler à l'application. Lorsqu'on intègre des applications collaboratives client/serveur ou multiserveurs, on doit coupler le (les) *Server Wrapper*(*s*) à son (ses) serveur(s). Dans le cas d'applications P2P, le *P2P Wrapper* est couplé à chaque application paire.

Le processus de couplage des *Wrappers* aux applications est partiellement automatisé. Si d'un côté nous avons une génération automatique des *Wrappers*, le couplage de ces *Wrappers* doit être fait manuellement. Cet effort de développement consiste à implémenter le lien entre les différentes fonctions de l'API de l'application collaborative et les méthodes définies dans l'*Application Interface*, tel que nous l'illustrons dans la Figure V.3. Remarquons que toutes les fonctions disponibles dans l'API d'une application ne correspondent pas nécessairement à des méthodes de l'*Application Interface*. Vu que l'API d'une application peut être très générale, elle peut définir des fonctionnalités qu'il peut ne pas être intéressant d'intégrer à LEICA. C'est bien sûr au développeur de choisir les fonctions de l'API de l'application à utiliser. Ce choix est exprimé lors de la création du fichier d'attributs et d'API (sous la forme de la spécification des événements que l'application est capable de notifier et des requêtes d'actions qu'elle est capable de traiter).

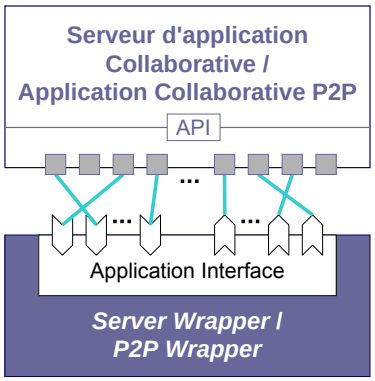

Figure V.3 Le couplage d'un Wrapper au serveur d'une application collaborative ou à une application P2P

L'automatisation complète du processus de couplage des *Wrappers* à ces applications pourrait être envisageable. Pour cela, on devrait être capable de spécifier dans le fichier d'attributs et d'API, au moyen de balises XML, les liaisons entre les fonctions de

l'API de l'application et les méthodes de l'*Application Interface*. Cela n'impliquerait cependant pas une réduction de l'effort de développement, car le travail de programmation se transformerait en un travail de description de méta-données. De plus, nous devrions définir un format XML, pour le fichier d'attributs et d'API, beaucoup plus complexe, ce qui augmenterait d'autant le temps d'apprentissage pour qu'un développeur puisse spécifier ce fichier.

V.2.1.3. Enregistrement d'une application

Pour s'enregistrer auprès de LEICA, un *Wrapper* publie ses services dans un *UDDI Registry* privé. Ce registre sert donc de répertoire des applications intégrées à LEICA, dans lequel nous trouvons les informations nécessaires pour contacter les *Wrappers*. En pratique, chaque *Wrapper* doit simplement publier l'identificateur de l'application et l'URL d'accès à son interface de services Web. Puisque les *Wrappers* ont tous la même interface de services, il n'est pas nécessaire que chaque *Wrapper* publie dans le registre la description de ses services (sous la forme d'un fichier WSDL).

Dans le cas d'une application client-serveur, l'enregistrement se fait automatiquement lorsqu'on exécute le serveur [1]. Dans le cas d'une application multiserveurs, le *Wrapper* de chaque serveur, lorsqu'il sera exécuté, essayera de s'enregistrer auprès de l' *UDDI Registry* privé, mais seul le premier y réussira. Celui-ci sera donc désigné serveur *Master* (ou maître). Les autres serveurs, suite à un échec d'enregistrement, contacteront directement le *Wrapper* du serveur maître pour que celui-ci soit au courrant des autres serveurs.

Pour les applications P2P, l'enregistrement est réalisé par l'intermédiaire du *P2P Proxy*. Il est nécessaire d'exécuter l'application une première fois pour qu'elle contacte le *P2P Proxy* et lui transmette toute l'information dont il aura besoin pour "jouer le rôle" de service Web pour cette application. Cette information correspond en fait aux contenus du ficher de données spécifiques et du fichier d'attributs et d'API (utilisés lors de la création du *P2P Wrapper*). Chaque fois qu'une application P2P envoie une requête du type *registry* au *P2P Proxy*, celui-ci crée et déploie dynamiquement une interface de service Web pour représenter cette application. Il enregistre ensuite cette nouvelle interface de service Web auprès du *UDDI Registry* privé.

Dans la Figure V.4, nous illustrons le processus d'enregistrement pour chaque type d'application.

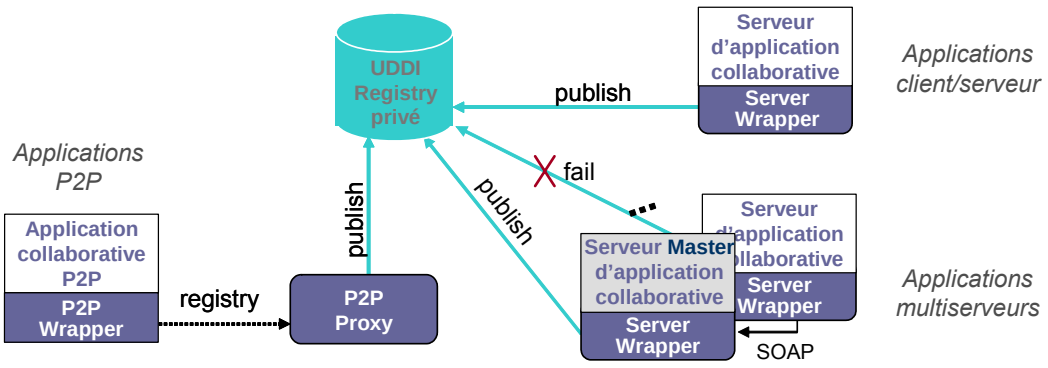

Figure V.4 Enregistrement d'applications collaboratives

[1] Une application non-collaborative se comportera tel qu'un serveur d'une application client-serveur.

V.2.2. Configuration d'une *SuperSession*

Le *Session Configuration Service* est le module de LEICA en charge de gérer la création de nouvelles *SuperSessions*. Ce module caractérise un service Web accessible par l'intermédiaire d'une application Web qui permet aux utilisateurs de créer et configurer des *SuperSessions*. Lorsqu'on accède à l'application Web pour créer une *SuperSession*, cette application contacte le *Session Configuration Service* pour démarrer le processus de configuration.

V.2.2.1. Configuration du *GSMinfo* et découverte des applications intégrées à LEICA

Comme nous l'avons décrit dans la section IV.4.1.1, la première étape du processus de configuration consiste à fournir l'ensemble des informations de gestion correspondant au *GSMinfo* de la *SuperSession* (illustré par l'étape (1) de la Figure V.5). L'administrateur des *SuperSessions*[1] doit à ce moment définir le nom de la *SuperSession*, les rôles généraux, sa planification, *etc*.

Comme cela est illustré dans l'étape (2) de la Figure V.5, pendant que l'administrateur de la *SuperSession* spécifie les informations de gestion, le *Session Configuration Service* cherche à découvrir quelles sont les applications qui ont été intégrées à LEICA en contactant le *UDDI Registry* privé. Il obtient en réponse la liste de ces applications, ainsi que les URLs associées aux interfaces de services Web rendues disponibles par les *Wrappers* et le *P2P Proxy*.

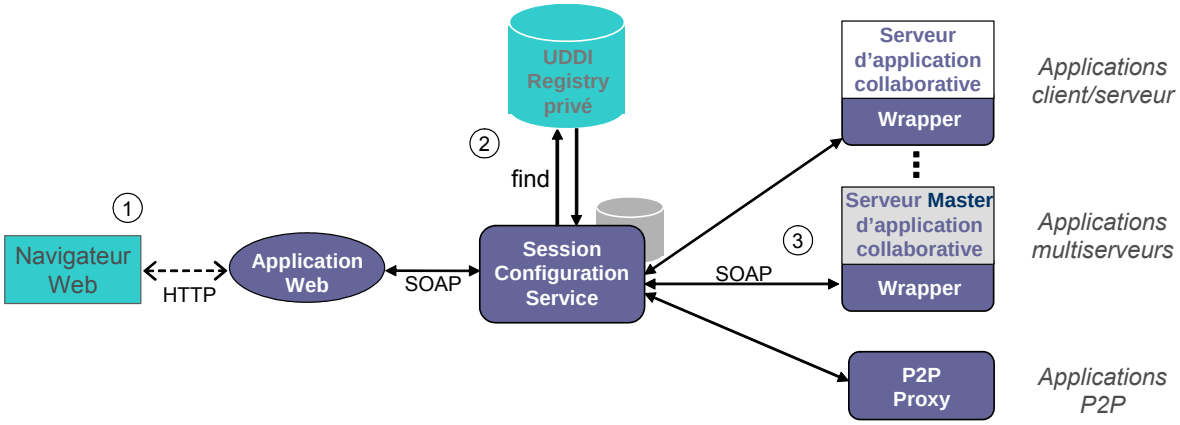

Figure V.5 Création d'une nouvelle SuperSession

Ensuite, comme cela est illustré dans l'étape (3) de la Figure V.5, le *Session Configuration Service* contacte chacun des *Wrappers* (dans le cas d'une application multiserveurs, seul le *Wrapper* du serveur maître est contacté) et le *P2P Proxy* en leurs demandant :

(i) les attributs caractérisant chaque application ;

(ii) les données nécessaires à chaque application collaborative pour créer des sessions spécifiques[2] ;

[1] L'administrateur correspond à l'utilisateur disposant des droits d'accès à l'application Web.
[2] Cette information n'est pas demandée aux applications non-collaboratives.

(iii) les types d'événements que les applications peuvent notifier et les types de requête que les applications peuvent exécuter (c'est-à-dire, l'API événements/ actions et l'API de gestion de l'application).

Les informations envoyées par les *Wrappers* et par le *P2P Proxy* sont temporairement sauvegardées dans le *Session Configuration Service*.

V.2.2.2. Choix et configuration des applications collaboratives

Dès que l'administrateur de la *SuperSession* a terminé de configurer les informations de gestion, il lui est demandé de choisir parmi l'ensemble des application disponibles (c'est-à-dire, l'ensemble des applications intégrées à LEICA) celles qu'il désire utiliser dans la *SuperSession*. L'administrateur doit alors configurer chacune des applications collaborative sélectionnées [1]. Cette étape du processus de configuration consiste donc à fournir l'ensemble des informations correspondant à l'*IAinfo* de la *SuperSession*. Comme nous l'avons spécifié dans la section IV.4.1.2, l'administrateur doit à ce moment spécifier les informations nécessaires pour créer des sessions spécifiques pour chaque application collaborative, et au moins une session spécifique par application collaborative choisie doit être définie.

Afin de connaître les types de données dont chaque application collaborative sélectionnée a besoin pour créer des sessions spécifiques, l'application Web demande cette information au *Session Configuration Service*. Par la suite, l'application Web construit dynamiquement des formulaires permettant à l'administrateur de la *SuperSession* de spécifier les informations nécessaires pour créer des sessions spécifiques. Par exemple, supposons qu'une application de vidéoconférence multicast soit intégrée à LEICA et que l'administrateur choisisse de la mettre en œuvre dans la *SuperSession*. Ce sera à ce moment là qu'il lui sera demandé de spécifier l'adresse IP multicast associée à une session spécifique de cette application de vidéoconférence.

V.2.2.3. Spécification de la politique de collaboration

La dernière étape du processus de configuration concerne la spécification de la politique de collaboration de la *SuperSession*. Cette spécification est réalisée en utilisant un éditeur graphique de règles politiques contenu dans l'application Web.

Avant le lancement de l'éditeur de règles politiques, l'application Web demande au *Session Configuration Service* l'API événements/actions de chaque application sélectionnée par l'administrateur de la *SuperSession*. Cela permettra à l'application Web de configurer dynamiquement l'éditeur de règles de façon à qu'il connaisse tous les types d'événements et d'actions qui pourront être utilisés dans la composition des règles.

La fin de la spécification de la politique de collaboration caractérise la fin de la configuration de la *SuperSession*. L'application Web engendre alors un fichier de configuration[2] et l'envoie au *Session Configuration Service*. Ce dernier le sauvegarde dans une base de données locale.

[1] Dans le cas d'applications non-collaboratives, on a uniquement besoin de sélectionner les applications.
[2] Au format XML.

V.2.3. Exécution d'une *SuperSession*

Une fois qu'une *SuperSession* a été créée, on peut lancer son exécution. Au cours de l'exécution d'une *SuperSession*, des utilisateurs peuvent la rejoindre afin de réaliser l'activité collaborative désirée.

V.2.3.1. Démarrage d'une *SuperSession*

Le *Session Configuration Service* est également en charge de démarrer les *SuperSessions*. Le lancement peut se faire automatiquement, s'il a été planifié ainsi dans le fichier de configuration de la *SuperSession*, ou manuellement. Le démarrage manuel doit se faire par l'intermédiaire de l'application Web qui donne accès au *Session Configuration Service*. La Figure V.6 illustre les étapes du processus de démarrage d'une *SuperSession*.

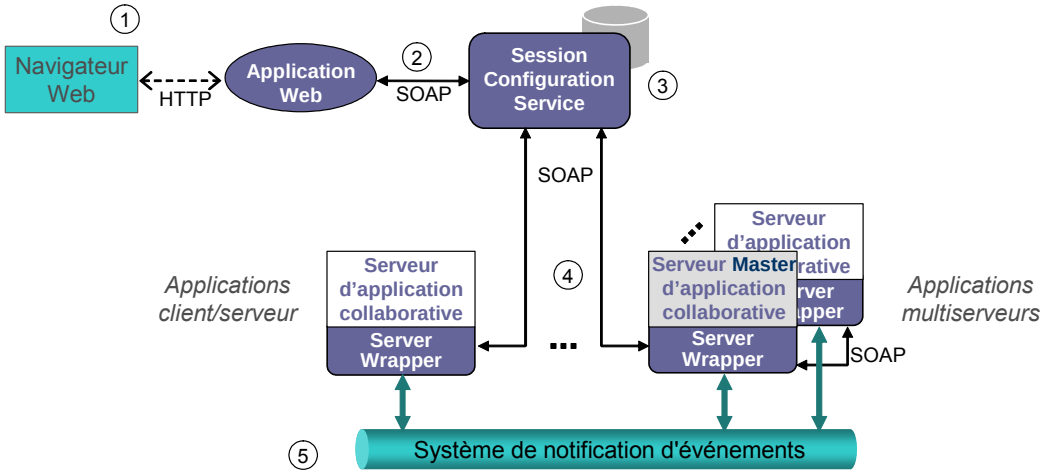

Figure V.6 Démarrage d'une SuperSession

1. L'administrateur accède à l'application Web et choisit parmi les *SuperSessions* déjà créés celle qu'il désire démarrer.
2. L'application Web fait une requête au *Session Configuration Service*.
3. Le *Session Configuration Service* récupère de la base de données locale le fichier de configuration de la *SuperSession* et l'analyse afin d'identifier les applications collaboratives qui seront utilisées dans cette *SuperSession*.
4. Le *Session Configuration Service* contacte les *Server Wrappers* concernés pour leur annoncer leur participation à la *SuperSession* en leurs envoyant les informations de configuration de la *SuperSession*. S'appuyant sur les informations reçues, chaque *Server Wrapper* identifie les sessions spécifiques qui doivent être créées et demande au serveur associé de créer ces sessions[1]. Remarquons que dans le cas d'applications multiserveurs, seul le *Server Wrapper* du serveur maître est initialement contacté, et ce dernier est en charge de contacter les autres *Server Wrappers* et de leur relayer les informations de configuration de la *SuperSession* (ceci en excluant les informations concernant les sessions spécifiques qui ne doivent être traitées que par le *Server Wrapper* du serveur maître). Les *Server Wrappers* construisent alors dynamiquement des sous-modules pour gérer l'état et exécuter la politique de collaboration de la *SuperSession*.

[1] S'il s'agit d'une session spécifique planifiée, la demande n'est réalisée qu'à l'instant spécifié.

5. Finalement, les *Server Wrappers* sont interconnectés au moyen d'un système de notification d'événements (les services Web ne sont plus utilisés à ce niveau de communication comme nous l'avons justifié auparavant).

Le système de notification d'événements utilisé par LEICA s'appuie sur le paradigme *publish/subscribe* [Eugster-03], car un tel schéma d'interactions est très bien adapté à des environnements faiblement couplés. En général, les *subscribers* (ou "souscripteurs") s'abonnent à des types d'événement particuliers, exprimant ainsi leur intérêt pour ces types d'événement. Ils reçoivent ensuite de manière asynchrone les événements qui correspondent à ces types, quelle que soit la source de ces événements, c'est-à-dire, indépendamment des *publishers* (ou "producteurs"). Ce paradigme favorise le découplage car il propose des modes de communication très souples : les *Wrappers* ne communiquent pas directement entre eux, mais s'échangent uniquement des messages. En plus, le fait que les d'abonnement soient gérés dynamiquement rend ce type d'architecture très flexible et approprié pour notre système de notification d'événements.

L'arrêt d'une *SuperSession* se fait également par l'application Web d'accès au *Session Configuration Service* (ou automatiquement dans le cas où il est programmé). Le *Session Configuration Service* doit simplement demander à un des *Wrappers* (parmi ceux impliqués dans la *SuperSession*) de publier une notification spéciale dans le système de notification d'événements indiquant la fin de la *SuperSession*. Ensuite, chaque *Wrapper* est chargé de demander à l'application collaborative associée d'arrêter toutes les sessions spécifiques créées pour cette *SuperSession*, tout en traitant la déconnexion des utilisateurs.

V.2.3.2. Connexion des utilisateurs à la *SuperSession*

Pour se connecter à une *SuperSession*, les utilisateurs doivent exécuter l'application *LClient*. La Figure V.7 illustre les étapes exécutées lors de la connexion d'un nouvel utilisateur.

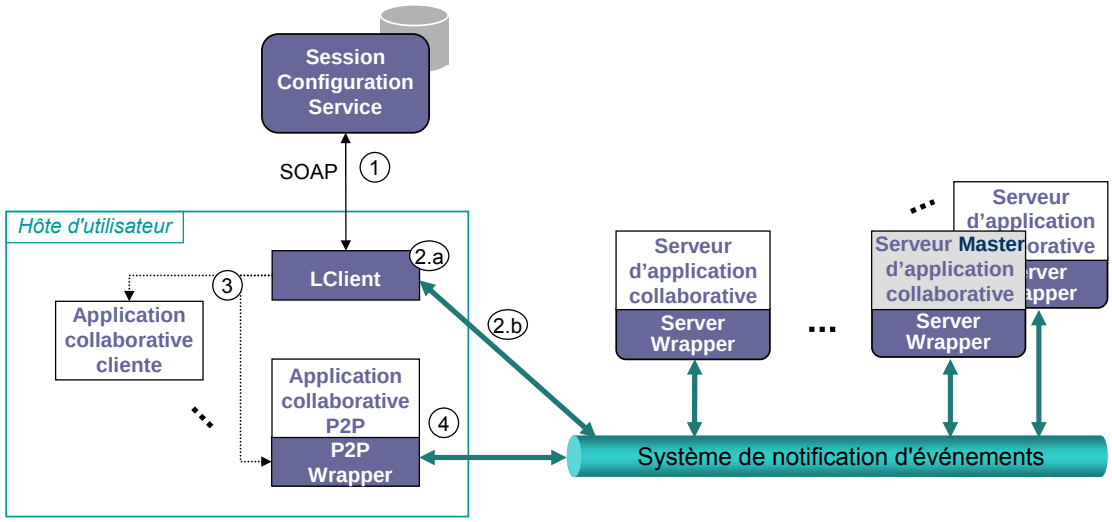

Figure V.7 Connexion d'un utilisateur à une SuperSession

1. *LClient* contacte le *Session Configuration Service* et reçoit en retour la liste de *SuperSessions* en cours d'exécution. L'utilisateur choisit de se connecter à une de ces *SuperSessions*, et le *LClient* récupère alors les informations de configuration (c'est-à-dire, le *GSMinfo*, l'*IAinfo*, et la politique de collaboration de la *SuperSession*).

2. Rappelons que, dans la section IV.4.2.1, nous avons dit que LEICA doit traiter des actions de l'API de gestion[1] des applications collaboratives (dans le cas où ces applications ne soient pas capables de le faire). C'est donc le *LClient* qui se charge d'exécuter ces actions. Ainsi, en étant responsable de traiter les requêtes de type "CONNECT", *LClient* permet de lancer localement les applications P2P et les applications clientes. De plus, *LClient* est en charge de la partie de l'API de gestion qui concerne des événements et des actions de gestion relatifs à LEICA[2].
   a. *LClient* construit un sous-module pour gérer l'état de la *SuperSession*, et un autre pour exécuter la politique de collaboration de la *SuperSession*. Ce dernier sous-module n'est donc concerné que par des règles politiques spécifiant des actions définies dans l'API de gestion[3]. *LClient* identifie dans ces règles quels sont les types d'événement qui peuvent les sensibiliser.
   b. *LClient* se connecte au système de notification d'événements et s'abonne à ces types d'événement. Ensuite, pour notifier la connexion de l'utilisateur qu'il représente dans la *SuperSession*, *LClient* envoie une notification d'événement de type "CONNECT", avec "LEICA" comme source de l'événement.
3. Au fur et à mesure que des événements sont notifiés, les règles politiques sont analysées. Lorsqu'elles sont sensibilisées et tirées, les actions de type "CONNECT" sont traitées par le *LClient* ce qui conduit à lancer l'exécution d'applications clientes ou P2P.
4. Dans le cas d'une application P2P, lorsqu'elle est lancée par un *LClient*, son *P2P Wrapper* reçoit les informations de configuration de la *SuperSession*. Ensuite, (de manière similaire à un *Server Wrapper* lors du démarrage d'une *SuperSession*), le *P2P Wrapper* construit dynamiquement des sous-modules pour gérer l'état et exécuter la politique de collaboration de la *SuperSession*. Il se connecte finalement au système de notification d'événements et exécute le même processus *publish*/*subscribe* que celui exécuté par le *Server Wrapper*, décrit dans la prochaine section.

V.2.3.3. Les notifications d'événement et l'exécution de la politique de collaboration

Au cours de l'exécution d'une *SuperSession*, chaque *Wrapper* est en charge de notifier les événements correspondant aux activités des utilisateurs connectés dans des sessions spécifiques de l'application à laquelle il est associé. Autrement dit, les serveurs et les applications P2P, par le biais de l'API de l'application, repassent aux *Wrappers* des notifications d'événement représentant "tout ce que se passe" à l'intérieur d'une session spécifique, et ce dernier les publie via le système de notification d'événements.

L'intérêt de ces notifications d'événements est que certaines peuvent sensibiliser des règles politiques définies pour cette *SuperSession*. Ainsi, le *Wrapper* a uniquement besoin de publier des événements qui peuvent sensibiliser des règles politiques, c'est-à-dire, des notifications d'événement qui sont utilisées dans les règles politiques. La politique de

[1] Des actions du type "CONNECT" et "DISCONNECT".
[2] Comme décrit dans la section IV.4.2.1, cette partie de l'API de gestion présente les événements de type "CONNECT" et "DISCONNECT" (pour notifier la connexion/déconnexion d'un utilisateur dans une *SuperSession*) et l'action de type "DISCONNECT" (pour forcer la déconnexion d'un utilisateur d'une *SuperSession*).
[3] Des actions de type "CONNECT" (avec pour cible (i) des applications P2P ou (ii) des applications client/serveur dont les serveurs ne sont pas capables des les traiter) et "DISCONNECT" (avec pour cible (i) LEICA ; (ii) des applications P2P ; ou (iii) des applications client/serveur dont les serveurs ne sont pas capables de les traiter).

collaboration impose donc de mettre en place des filtres pour les notifications d'événements. Ainsi, en faisant appel au sous-module responsable de l'exécution de la politique de collaboration de la *SuperSession*, le *Wrapper* sait quels types d'événement il doit effectivement publier.

Au cours d'une *SuperSession*, en plus de publier des notifications d'événement, chaque *Wrapper* doit parallèlement exécuter l'ensemble des règles définies dans la politique de collaboration de la *SuperSession*. Un *Wrapper* n'est cependant concerné que par les règles qui définissent des actions à exécuter par l'application à laquelle il est associé. Autrement dit, le *Wrapper* ne doit exécuter que les règles politiques composées des *Actions* dont le paramètre "to" est l'identificateur de l'application correspondant à ce *Wrapper*. Ainsi, le *Wrapper* n'est intéressé que par des événements qui peuvent sensibiliser ces règles politiques. Il fait donc appel à son sous-module responsable de l'exécution de la politique de collaboration pour connaître les types d'événements auxquels il doit s'abonner dans le système de notification d'événements.

A titre d'exemple, considérons la politique de collaboration illustrée dans la Figure V.8. Le *Wrapper* de l'application collaborative "CA1" (supposons que ce soit une application client/serveur) ne doit publier que des événements de type "T1" et "T3". De plus, puisque seules les règles 2 et 4 définissent des actions qui doivent être exécutées par l'application "CA1", seules ces règles doivent être exécutées par le *Wrapper*.

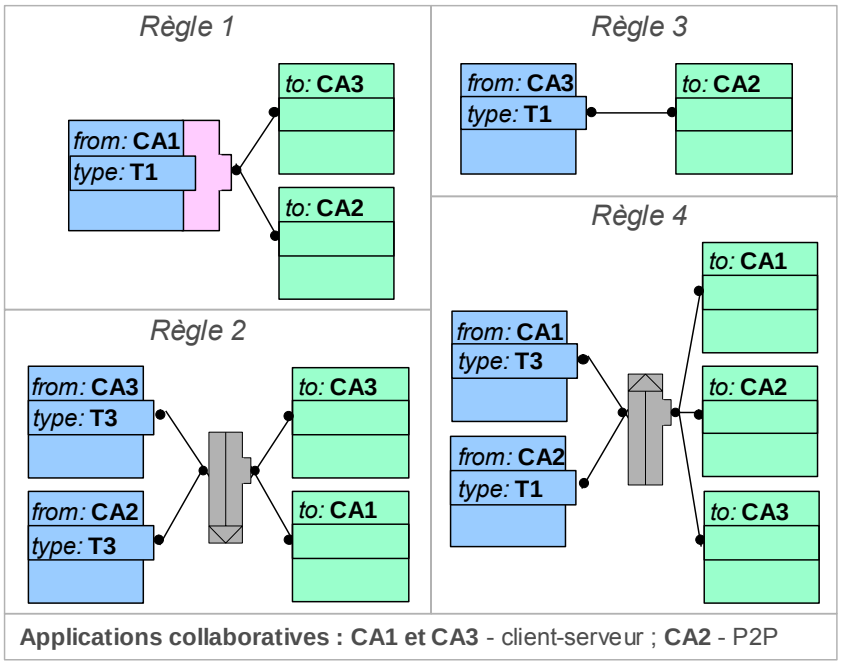

Figure V.8 Exemple simplifié de politique de collaboration d'une SuperSession

Remarquons que LEICA dispose d'un système reparti d'exécution de la politique de collaboration d'une *SuperSession*. En considérant l'exemple de la Figure V.8, la règle 4 doit être exécutée par les trois applications collaborative en parallèle. Lorsque "CA1" notifiera l'événement de type "T3", la règle sera alors sensibilisée dans chaque *Wrapper* des trois applications. Ensuite, chaque application devra traiter la requête correspondante.

V.2.3.4. Gestion de l'état global d'une *SuperSession*

Chaque *Wrapper* et chaque *LClient* doivent maintenir une version locale de l'état global de la *SuperSession*. Ceci est nécessaire, car les règles politiques peuvent dépendre de l'état global de la *SuperSession*.

Les composants de l'état d'une *SuperSession* sont définis selon le modèle de *SuperSession* décrit dans la section IV.3.1. Nous pouvons classer ces composants d'état en deux catégories, (i) statique et (ii) dynamique. Les composants statiques correspondent aux entités statiques d'une *SuperSession*, c'est-à-dire, les éléments qui sont prédéfinis lors de la configuration d'une *SuperSession* : l'ensemble des applications utilisées, leurs sessions spécifiques, et l'ensemble des rôles généraux. Les composants dynamiques représentent les entités dynamiques d'une *SuperSession*, c'est-à-dire, tous les autres éléments qui peuvent varier au cours du temps. Notamment, les utilisateurs connectés et les ressources accédées.

Lorsque le sous-module de gestion d'état est créé par le *Wrapper*, il engendre tout d'abord les composants d'état statiques. Ensuite, l'état de la *SuperSession* évolue en fonction des événements survenus. Plus précisément, la création et la modification des composants dynamiques de l'état se fait conformément aux notifications d'événement associées à l'API de gestion. Nous voyons donc l'intérêt à contraindre les applications collaboratives à être capables de notifier les événements de l'API de gestion.

Un des *Wrappers* utilisés lors de l'exécution d'une *SuperSession* est automatiquement désigné en tant que *State Manager* (ou "Gestionnaire de l'état global"). Celui maintient une copie "maître" de l'état de la *SuperSession* et à chaque nouvelle connexion au système de notification d'événement (soit d'un *Wrapper*, soit d'un *LClient*) il envoie au nouvel entrant une notification d'événement de type "STATE". Ce message contient l'état actuel des composants dynamiques de l'état global de la *SuperSession*.

Remarquons qu'une *SuperSession* est exécutée sur un système reparti et la construction de son état global dépend des informations véhiculées par des notifications d'événement (*i.e.* des messages). Bien évidement, les délais de transmission de ces messages peuvent varier, et par conséquent, deux sites (*Wrappers*) différents peuvent avoir des informations différentes selon l'ordre des événements. Ainsi, ces sites peuvent avoir des informations différentes sur l'état global de la *SuperSession*, ou même, dans certaines situations, des incohérences peuvent arriver dans la gestion de cet état global.

En outre, la cohérence de l'exécution de la politique de collaboration de la *SuperSession* est également affectée par la distribution des notifications d'événement, d'autant plus que l'exécution des règles politiques est elle-même un processus exécuté de manière repartie.

Nous aborderons dans le prochain paragraphe les incohérences potentielles de l'exécution répartie des *SuperSessions*.

V.2.4. Les incohérences potentielles dues à la répartition

Nous abordons dans ce paragraphe des incohérences relatives à l'exécution d'une *SuperSession* en l'absence de contraintes temporelles, comme celles que nous venons d'introduire dans le paragraphe précédent, ainsi que des incohérences en présence de contraintes temporelles.

V.2.4.1. La gestion d'un état global cohérent et l'exécution repartie de la politique de collaboration

Le calcul d'un état global dans un système reparti s'appuyant sur des notifications d'événements, où les messages véhiculant ces notifications sont susceptible d'avoir différents délais de transmissions, est en fait l'un des paradigmes des problèmes de contrôle reparti.

La première difficulté concerne le fait que les délais des messages véhiculant des informations nécessaires à la gestion de l'état global peuvent entraîner le non respect de la séquence des émissions du côté des récepteurs. Dans le cas spécifique de LEICA, un *Wrapper* peut avoir ainsi une information différente sur l'ordre des notifications d'événement par rapport à un autre *Wrapper*. Par exemple, considérons *a* et *b* des événement produits sur le même site, où *a* précède directement *b* (*a* étant la cause de *b*). Si les notifications de ces événements arrivent à un *Wrapper* (exécuté sur un autre site) dans un ordre inverse nous aurons une incohérence lors du calcul de l'état global.

Si nous considérons que des événements notifiés par des *Wrappers* distincts sont concurrents (ou causalement indépendants), il suffit que les notifications d'événement provenant d'une même source soient ordonnées pour permettre aux *Wrappers* de garder un état cohérent. Autrement, un ordonnancement total des événements, en utilisant par exemple des techniques d'ordonnancement par estampillage s'appuyant sur des horloges logiques [Lamport-78], permettrait de préserver un état cohérent dans toutes les situations.

Dans le cas spécifique de LEICA, cette gestion de l'état global entraîne un deuxième problème concernant le fait que chaque *Wrapper*, à un même instant donné, et malgré la garantie d'un ordonnancement des notifications, peut avoir des informations différentes sur l'état global de la *SuperSession*. Ceci peut avoir des conséquences directes sur l'exécution de la politique de collaboration étant donné qu'il s'agit d'une exécution repartie et que les règles peuvent poser des contraintes sur des composantes de l'état global de la *SuperSession*.

Revenons par exemple sur la règle 1 illustrée dans la Figure V.8. Nous savons que cette règle doit être gérée parallèlement par les *Wrappers* des applications CA2 et CA3. Supposons que le *Predicate* présent dans cette règle définisse un prédicat qui porte sur des composantes de l'état global de la *SuperSession*. Il est donc possible que, lorsque CA2 et CA3 reçoivent la notification de l'événement de type T1 provenant de CA1, l'évaluation du prédicat présente des valeurs différentes dans chacun des *Wrappers*. Par conséquent, la règle ne sera sensibilisée que dans un des *Wrappers* (là où il est évalué à vrai). Par contre, pour assurer la cohérence de l'exécution répartie d'une politique de collaboration, nous avons défini comme principe de base que : lorsqu'une même règle est gérée simultanément par différents *Wrappers*, dans le cas où cette règle est sensibilisée dans un *Wrapper*, elle doit l'être également dans tous les autres *Wrappers*. Ainsi, dans l'exemple que nous venons de décrire, nous pouvons violer ce principe de base.

En théorie, différentes solutions pourraient être employées pour garantir ce principe de base lors de l'exécution répartie de la politique de collaboration (vis-à-vis de la gestion de l'état global de la *SuperSession*). Une possibilité serait d'établir une "négociation" entre les différents *Wrappers* concernés par une même règle présentant des prédicats qui définissent des contraintes sur l'état global de la *SuperSession*. Lors de l'exécution d'une telle règle, les *Wrappers* devraient synchroniser les vues qu'ils ont de l'état global du système (par exemple, en employant des solutions telles que celles utilisées pour les

"*problèmes d'accord*" [Hurfin-98]) avant d'évaluer les prédicats qui portent sur des composantes de l'état global.

Néanmoins, dans la pratique, nous pouvons faire deux observations :

(i) la fréquence des notifications d'événement qui peuvent entraîner une modification de l'état global (c'est-à-dire, des événements définis dans l'API de gestion) n'est pas du même ordre de grandeur que la fréquence des toutes autres notifications.

(ii) LEICA s'appuie sur un système de notification d'événements assez performant, permettant aux différents *Wrappers* d'arriver à une même "vue" de l'état global de la *SuperSession* assez vite à chaque modification de cet état.

Nous pouvons ainsi considérer que les chances d'avoir un comportement incohérent, tel que nous venons de l'illustrer, ne justifient pas l'implémentation de mécanismes de synchronisation entre les *Wrappers*. En fait, la mis en œuvre de tels mécanismes pourrait compromettre la performance (ou le temps de réponse) du système d'exécution de la politique de collaboration (une performance qui nous a en réalité guidé vers la répartition de l'exécution des règles politiques).

V.2.4.2. Les incohérences en présence de contraintes temporelles

Si d'un côté la répartition de l'exécution des règles politiques permet de contribuer à la performance du sous-module d'exécution des règles politiques, d'un autre côté, comme nous avons déjà introduit, le système risque de présenter des incohérences.

Dans le cas où une règle n'est gérée que par un *Wrapper* (comme la règle 3 de la Figure V.8), il ne peut pas y avoir, par construction, d'incohérence. Si la règle est gérée par plus d'un *Wrapper*, mais est sensibilisée par une simple notification d'événement (comme la règle 1 de la Figure V.8), le principe de base est garanti dès que l'on dispose d'un système de notifications d'événements fiable[1]. Nous pouvons affirmer la même chose pour une règle composée de différentes notifications d'événement au moyen des *Earliests* (comme la règle 4 de la Figure V.8). Un *Earliest* définit en effet une composition d'événements sans imposer de corrélation entre eux.

Par contre, une composition de notifications d'événement au moyen de *Latests* peut poser un problème. Rappelons que chaque *Latest* est associé à un "temps d'attente" t_w définissant l'écart maximal accepté entre les instants de la première et de la dernière notification d'événement composées. Il existe ainsi une corrélation entre les événements vis-à-vis de l'intervalle de temps entre ces notifications.

Dans un système de notification d'événements réparti, deux récepteurs peuvent recevoir une même notification d'événement à des instants différents. Par conséquent, nous ne pouvons pas garantir que, suite à une séquence donnée de notifications d'événement, le délai observé entre ces notifications soit identique pour tous les récepteurs, et même la séquence peut être différente. En conséquence, des notifications peuvent sensibiliser une règle politique dans un *Wrapper*, mais ne pas la sensibiliser dans un autre *Wrapper* en charge de la même règle.

Considérons, par exemple, la règle 2 illustrée dans la Figure V.8. Supposons que "CA3" et "CA2" viennent de notifier chacune un événement de type "T3", et que le

[1] A l'exception du prédicat associé portant sur des composantes de l'état global de la *SuperSession*, comme nous l'avons précisé dans la section précédente.

Wrapper de l'application "CA1" reçoive ces notification aux instants t_1 et t_2, et que le *Wrapper* de l'application "CA3" les reçoive aux instants t_3 et t_4. Une situation d'incohérence se produira si $|t_1-t_2| \leq t_w < |t_3-t_4|$ ou si $|t_3-t_4| \leq t_w < |t_1-t_2|$; dans le premier cas la règle ne sera sensibilisée que dans le *Wrapper* de "CA1", et dans le deuxième cas que dans le *Wrapper* de "CA3".

Une solution envisageable pour traiter ce problème consisterait à ajouter une estampille temporelle à chaque notification d'événement lors de sa publication. Ainsi, pendant l'exécution d'une règle politique composée de notifications d'événement au moyen de *Latests*, il suffirait de considérer ces estampilles temporelles (dates de notification des événements) au lieu des instants d'arrivée des notifications. Néanmoins, pour que le délai entre les valeurs de deux estampilles temporelles produites dans des sites distincts soit correct, il est nécessaire que chaque site dispose d'une horloge physique et que ces horloges soient mutuellement synchronisées (par exemple, en utilisant des protocoles de synchronisation tel que NTP [Mills-92]).

V.2.4.3. Autres incohérences concernant l'exécution des règles politiques : la problématique des applications multiserveurs et P2P

LEICA prend pour hypothèse de base le fait que les notifications d'événement ne doivent pas être dupliquées. Cela signifie que l'occurrence d'un événement spécifique ne doit être notifiée qu'une et une seule fois. Autrement, le système d'exécution des règles peut présenter un comportement incohérent lorsque l'occurrence d'un événement donné est notifiée plus qu'une fois.

Pour les applications client-serveur, cette hypothèse de base est simple à garantir, alors que les applications multiserveurs ou P2P doivent disposer d'un mécanisme interne de consensus pour satisfaire cette contrainte. Ce mécanisme dépendra en fait de la politique employée par l'application collaborative pour gérer de manière répartie les éléments actifs (par exemple, des utilisateurs, des objets partagés, *etc.*) appartenant à une session spécifique.

En effet, dans la gestion d'objets repartis employée par la plus part des applications multiserveurs, chaque utilisateur ou chaque entité partagée n'est gérée que par un serveur à la fois. Dans ce cas, la politique à mettre en place est simple : chaque serveur ne doit repasser à son *Wrapper* que les événements engendrés par les utilisateurs ou entités dont il est responsable. Cette même logique peut être employée dans le cas d'applications P2P, où chaque application paire représente un utilisateur, et les entités partagées sont en général associées à un utilisateur, normalement celui qui l'a créé ou qui a été désigné comme propriétaire de l'entité.

Pour une requête d'exécution d'action, il faut également implémenter un mécanisme permettant aux différents serveurs ou applications P2P de savoir qui doit l'exécuter. Une manière d'obtenir un consensus consisterait à attribuer l'exécution d'une requête au serveur ou à l'application P2P qui est en charge de l'entité ou de l'utilisateur concerné par cette requête.

V.3. Modélisation de LEICA en UML

Dans le but de modéliser l'architecture de LEICA, nous avons choisi le langage UML (*Unified Modeling Language* ou "Langage de modélisation unifié"). Etant un

langage, et non pas une méthode, UML peut donc être utilisé sans remettre en cause les méthodes habituelles de conception. UML permet également de modéliser des systèmes quels que soient les langages et les plates-formes utilisés.

Plus spécifiquement, nous avons utilisé la version UML 2.0 supportée par le profil UML/SDL de l'outil *Tau G2* de *Telelogic* pour modéliser l'architecture et le comportement de LEICA. En fait, d'autres outils sont aujourd'hui disponibles : des logiciels libres, normalement moins puissants et performants, et des outils commerciaux, notamment *Rational Rose* [Rational], un des leaders mondiaux en outil de modélisation UML, mais aussi l'un des plus coûteux du marché. Par contre, *Tau G2*, au moment où nous avons commencé la conception de LEICA, était l'un de rares outils à prendre en compte la version 2.0 de UML. Cette nouvelle version de UML apporte plusieurs améliorations à la version 1.0, ainsi que plusieurs nouveaux concepts qui permettent de mieux supporter l'ingénierie système (en particulier, des modèles exécutables permettant aux architectes systèmes de "debugger" leur *design* avant le codage).

Nous allons introduire le langage avant de détailler la méthode employée pour modéliser et valider l'architecture de notre environnement d'intégration.

V.3.1. Introduction à UML 2.0

UML est un langage graphique destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, définir des architectures logicielles, concevoir des solutions et communiquer des points de vue. S'appuyant en particulier sur une approche orientée objets, UML est normalisé par l'OMG [OMG], un consortium de plus de 800 sociétés et universités actives dans le domaine des technologies objets.

Le standard UML 2.0 (qui propose un modèle de composants plus élaboré que celui de la version UML 1.0) permet de modéliser différentes vues des applications qui peuvent mettre en évidence différents aspects du logiciel à réaliser. Toutes les vues proposées par UML sont complémentaires les unes des autres, et elles sont représentées à l'aide de différents diagrammes (13 au total):

- La vue fonctionnelle cherche à capturer les interactions entre les différents acteurs/utilisateurs et le système, sous forme d'objectifs à atteindre (à l'aide de *Diagrammes de cas d'utilisation*) et sous forme de scénarios d'interaction typiques (à l'aide de *Diagrammes de séquences*). En outre, nous avons les *Diagrammes de communication*, qui contiennent la même information que les diagrammes de séquences mais qui insistent plus sur l'aspect structurel des objets en interaction que sur l'aspect chronologique.

- La vue structurelle, ou statique, réunit les *Diagrammes de classes*, les *Diagrammes d'objets* et les *Diagrammes de packages*. Les premiers tentent d'identifier les types d'objets constituant le programme, leurs attributs, opérations et méthodes, ainsi que les liens ou associations qui les unissent. Les deuxièmes permettent d'illustrer des configurations particulières d'objets en complément aux diagrammes de classes. Les troisièmes servent à regrouper les classes fortement liées entre elles en des composants les plus autonomes possibles[1]. De plus, nous disposons des *Diagrammes de composants*, qui permettent de représenter la hiérarchie d'un projet regroupant des objets en composants logiciels, et les *Diagrammes de structures composites*, qui servent à montrer la structure interne

[1] A l'intérieur de chaque *package*, on trouve des diagrammes de classes.

d'un composant (typiquement une classe) y compris ses points d'interaction avec le reste du système. Les derniers diagrammes structurels sont les *Diagrammes de déploiement*, qui relèvent plus spécifiquement de l'implémentation, en indiquant sur quelle architecture matérielle seront déployés les différents processus qui réalisent l'application.

- La vue dynamique, à l'aide notamment de *Diagrammes d'états*, vise à décrire l'évolution des objets complexes du système tout au long de leur cycle de vie : les objets voient leurs changement d'états guidés par leurs interactions avec les autres objets. Les *Diagrammes temporels* (ou *Chronogrammes*) constituent une alternative aux diagrammes d'états, et ils sont conçus pour représenter l'évolution au cours du temps de la valeur ou de l'état d'un ou plusieurs éléments. Nous avons également les *Diagramme d'activités* qui correspondent à une sorte d'organigramme permettant de modéliser des activités qui se déroulent en parallèle les unes par rapport aux autres. En général, les diagrammes d'états à eux seuls ne permettent pas de faire apparaître les problèmes spécifiques posés par la synchronisation de processus concurrents. Finalement, nous avons les *Diagrammes d'interaction globaux*. Il s'agit d'une nouvelle forme de diagrammes d'activités dans lesquels les nœuds sont des diagrammes d'interaction[1].

V.3.2. Méthode de modélisation utilisée

Classiquement, la modélisation démarre par une phase d'analyse où nous réalisons une formalisation précoce des besoins des utilisateurs du système au moyen de diagrammes de cas d'utilisation. Ces diagrammes font un bilan à haut niveau des services de base offerts par le système et des interactions avec les utilisateurs du système.

Pour combler le fossé entre la vue fonctionnelle, initialement donnée par les diagrammes de cas d'utilisation, et la vue structurelle nous identifions les composant de base du système en construisant plusieurs diagrammes de séquences. Ces diagrammes représentent des scénarios nominaux et dégradés du comportement du système. Ils sont accompagnés d'une liste précise des hypothèses sous lesquelles la modélisation est effectuée. Les scénarios identifiés dans cette étape serviront ainsi de base à la construction de la structure du système.

La phase d'analyse est suivie d'une phase de conception qui précise la vue structurelle du système. Dans cette phase des diagrammes de classes et de structures composites sont spécifiés pour décrire la structure statique du système.

[1] Par diagramme d'interactions on désigne la catégorie de diagrammes constituée des diagrammes de séquences, de communications, temporels et des diagrammes d'interactions globaux.

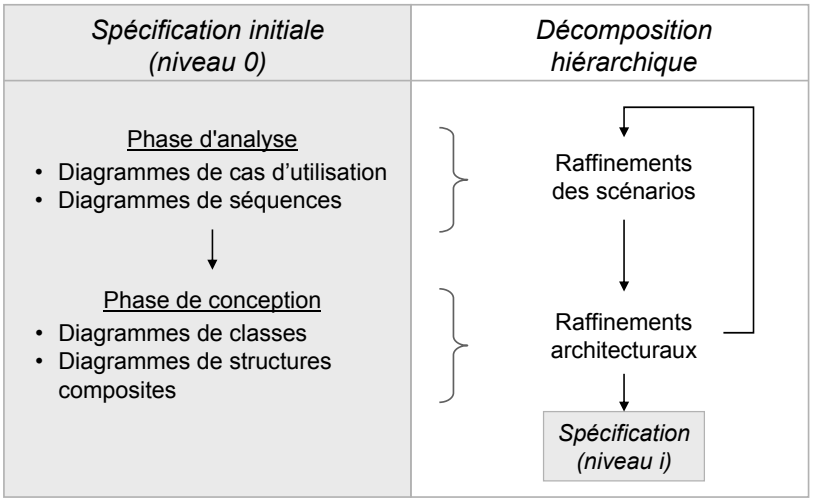

Figure V.9 Approche top-down de spécification du système

Dans le but de réaliser une construction incrémentale de la structure du système par levée progressive des hypothèses restrictives, nous avons utilisé une méthode de description *top-down* classique, en suivant l'approche illustrée dans la Figure V.9. Nous avons ainsi commencé par une spécification initiale composée des diagrammes de plus haut niveau ("niveau 0"), qui identifient simplement les utilisateurs et le système. Ensuite, nous avons réalisé une décomposition hiérarchique en exécutant une analyse itérative par raffinements successifs des diagrammes. Premièrement des raffinements au niveau des diagrammes de cas d'utilisation et des diagrammes de séquences sont appliqués pour décrire les interactions entre des sous-systèmes. Ensuite les raffinements architecturaux correspondants sont réalisés sur les diagrammes de classes et les diagrammes de structures composites. De chaque itération résulte une nouvelle version de la spécification du système représentant ainsi un nouveau niveau de raffinement. Nous avons poursuivi cette approche jusqu'au niveau 3.

Après avoir détaillé la structure du système en différents niveaux de raffinement, nous avons défini les comportements des différents composants du système [1]. Ces comportements sont décrits par des machines à états finis à l'aide du langage SDL (*Specification Description Language*). Le langage SDL permet de spécifier un système sous la forme abstraite de plusieurs processus concurrents qui communiquent par l'intermédiaire de signaux. Ainsi le comportement de ces objets donne au modèle un caractère exécutable qui permet de mettre en œuvre une simulation au moyen de l'outil *Tau G2*. Cet outil produit des traces de simulation sous forme de diagrammes de séquences que l'on peut ensuite comparer manuellement aux diagrammes de séquences définis, à titre documentaire, lors de la phase d'analyse. Considérant la complexité du système, nous avons choisi de conduire une simulation intensive (dans laquelle on n'explore que partiellement, mais à la volée, les graphes de comportement), au lieu d'une simulation exhaustive (dans laquelle, partant d'un état initial, on explore toutes les séquences possibles de transitions).

Nous avons appliqué une méthode de validation informelle en utilisant des techniques d'observation. L'objectif est de fiabiliser la conception du système et d'assurer que l'implantation finale fonctionnera correctement sous des hypothèses environnementales bien identifiées. Si les traces de simulation obtenues ne correspondent

[1] A l'exception les *Wrappers* des applications multiserveurs et P2P.

pas aux diagrammes de séquences prédéfinis, cela implique que nous disposons d'un modèle de conception qui ne répond pas aux spécifications. Les résultats de la validation serviront à la correction du modèle d'analyse, à la génération d'un nouveau modèle de conception et ainsi de suite, jusqu'à l'obtention d'un modèle de conception conforme.

En conclusion, dans la modélisation de notre système, nous n'utiliserons pas tous les diagrammes introduits dans la section V.3.1, mais uniquement les diagrammes de cas d'utilisation, les diagrammes de séquences, les diagrammes de classes, les diagrammes de structures composites et les diagrammes d'états à la SDL .

V.3.3. Modélisation en UML de l'architecture de LEICA

Dans cette section nous allons présenter le processus de modélisation de l'architecture de LEICA, selon la méthode que nous venons d'expliquer.

V.3.3.1. L'analyse : les cas d'utilisation et les diagrammes de séquences de LEICA

La Figure V.10 présente le diagramme de cas d'utilisation général de LEICA. Dans ce diagramme, nous pouvons observer les différents "acteurs" (conformément à la notation d'UML), ou utilisateurs, et leurs interactions avec LEICA. En UML, les acteurs sont des entités externes qui ne font pas partie du système lui-même, et qui n'ont donc pas à être modélisées. Pour mieux illustrer les scénarios d'interactions, nous avons défini trois classes d'utilisateurs: (i) le *Developper* (celui qui réalise l'intégration d'une application à LEICA), le *SessionAdmin* (celui qui crée et configure de nouvelles *SuperSessions*, et les démarre), et le *User* (l'utilisateur qui rejoint une *SuperSession* en exécution). Remarquons qu'une partie des interactions illustrées par les cas d'utilisation correspond en fait aux trois activités principales nécessaires pour accomplir la mise en place de *SuperSessions* par l'environnement, tel que nous l'avons décrit dans la section V.2.

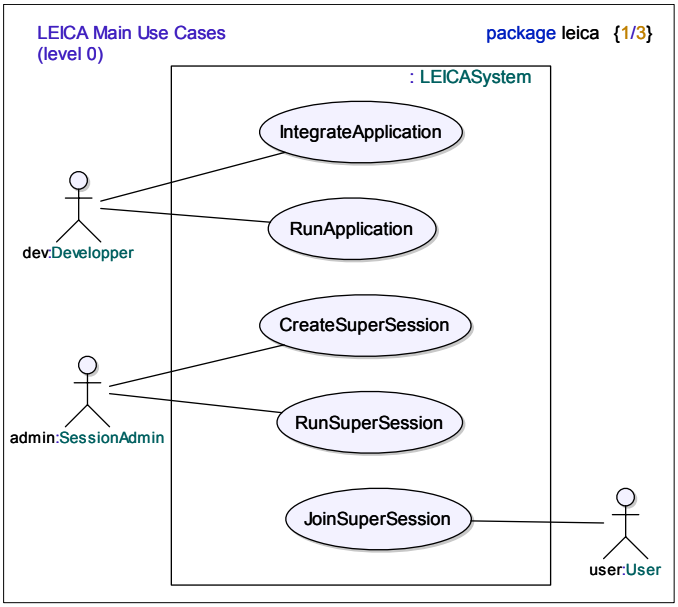

Figure V.10 Cas d'utilisation de LEICA

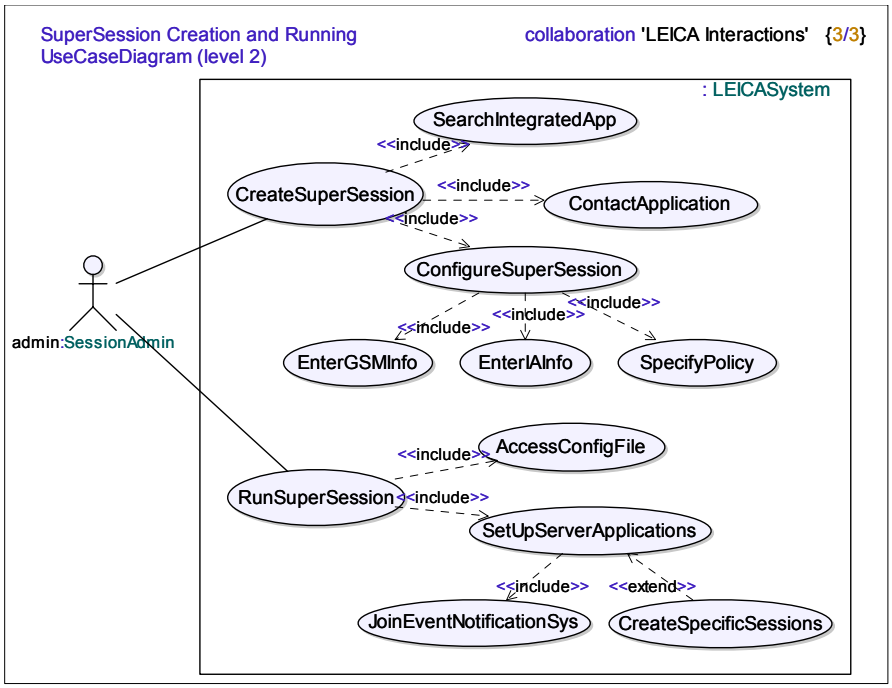

Figure V.11 Diagramme de cas d'utilisation pour configurer et démarrer une SuperSession

Après quelques raffinements de ce diagramme de cas d'utilisation général, nous sommes arrivés à des diagrammes plus détaillés. Dans la Figure V.11 nous présentons par exemple le diagramme de cas d'utilisation associé à la configuration et au démarrage d'une *SuperSession*.

Nous associons à chaque cas d'utilisation des diagrammes de séquences pour définir les interactions entre les différents composants (et sous-composants, quand nous avons des diagrammes plus raffinés) du système. Analysons, à titre d'exemple, le cas d'utilisation appelé "CreateSuperSession" spécifié dans la Figure V.11. Un des diagrammes de séquences défini pour ce cas d'utilisation concerne la création réussie d'une *SuperSession*, comme cela est illustré dans la Figure V.12, où nous notons quelques éléments de type *ref* (ou référence). Une référence peut être vue comme un pointeur ou un raccourci vers une autre interaction (cas d'utilisation) et/ou un autre diagramme de séquence existants. Cela permet de factoriser des parties de comportement qui peuvent être référencées dans plusieurs scénarii. Par exemple, la Figure V.13 illustre un diagramme de séquence, référencé dans le diagramme précédent, qui correspond au cas d'utilisation "ContactApplication" (qui apparaît également dans la Figure V.11), où le *Session Configuration Service* réquisitionne des informations soit d'un *Server Wrapper*, soit d'un *P2P Proxy*.

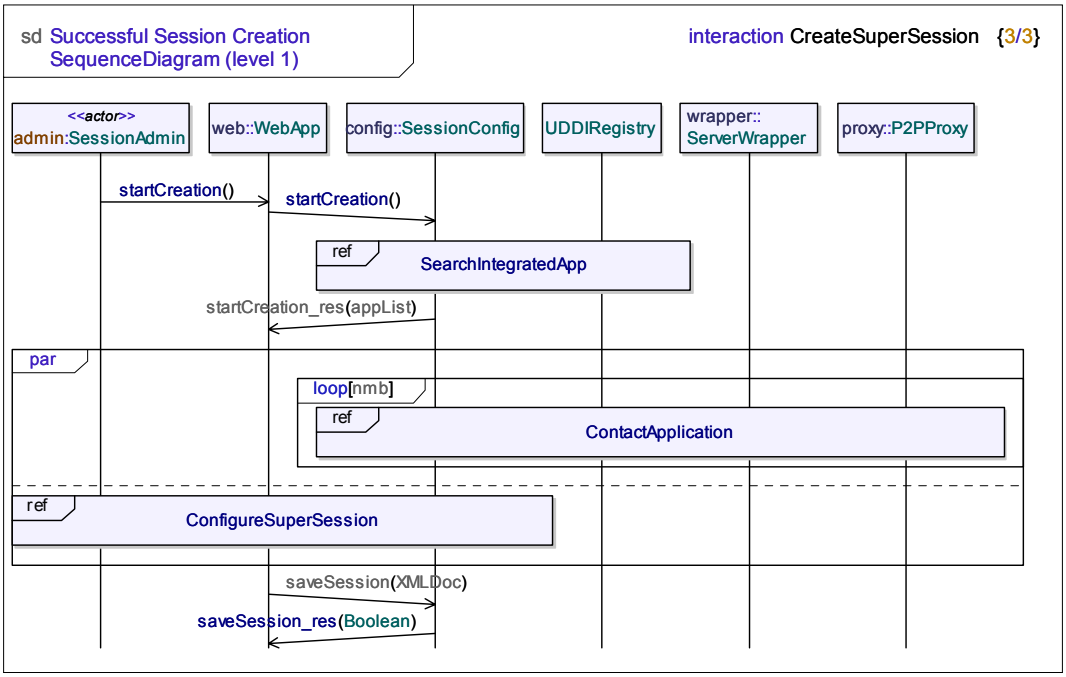

Figure V.12 Diagramme de séquences pour créer une SuperSession

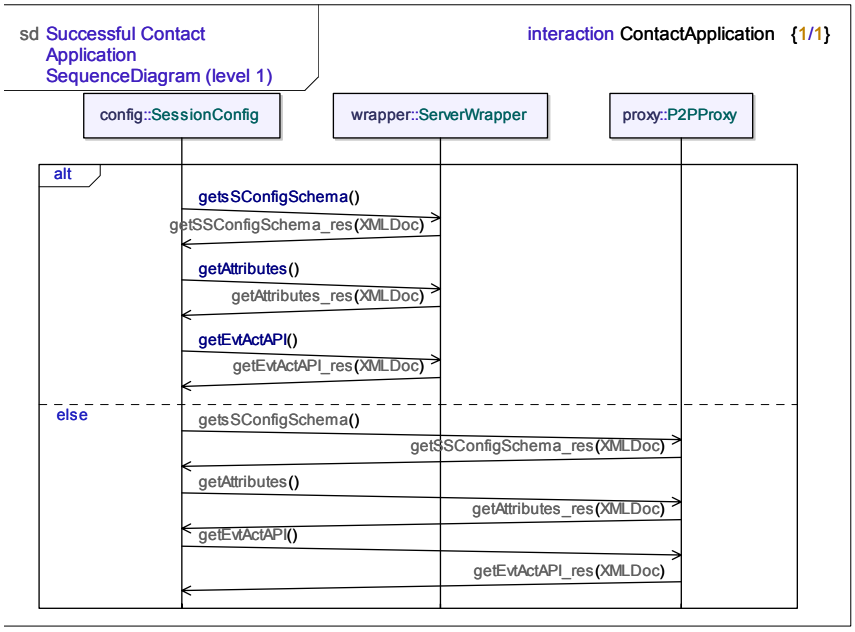

Figure V.13 Diagramme de séquences pour contacter des applications collaboratives

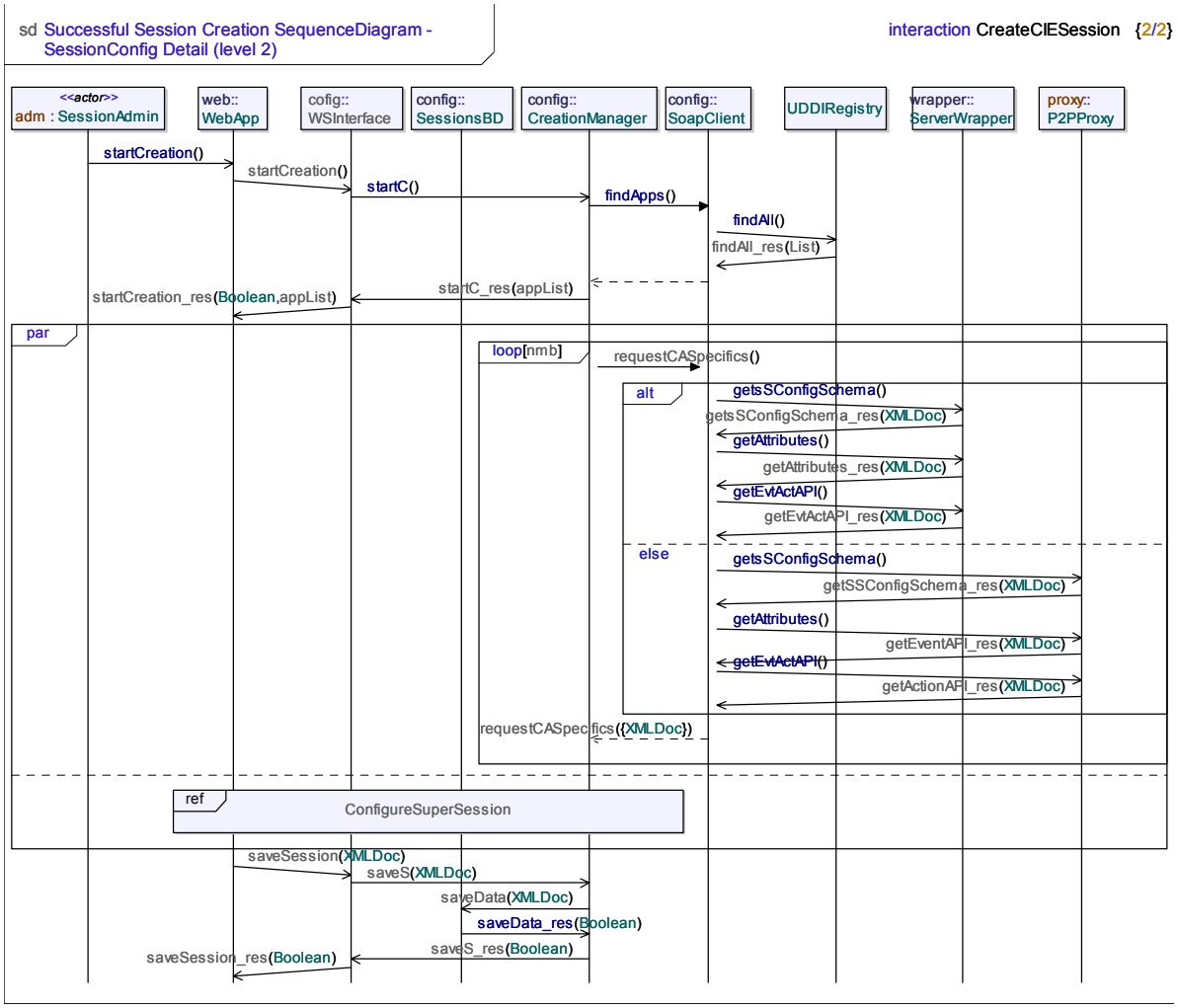

Figure V.14 Diagramme de séquences pour créer une SuperSession, détails sur les sous-composants du Session Configuration Service

Suite à la décomposition hiérarchique du modèle, les diagrammes de séquences ont été raffinés décrivant ainsi les interactions entre des sous-composants. Par exemple, nous illustrons dans la Figure V.14 le diagramme de séquences où nous voyons les interactions entre les sous-composants du *Session Configuration Service* lors de la création d'une *SuperSession*. Ce diagramme représente donc la même information que celle de la Figure V.12 mais de manière plus détaillée vis-à-vis de ce composant.

V.3.3.2. La conception : les diagrammes de classes et de structures composites

Le diagramme de classes de la Figure V.15 décrit la structure statique de LEICA[1]. La classe de plus haut niveau, appelée *LEICASystem*, est composée de plusieurs classes qui correspondent en fait aux éléments de l'architecture de LEICA décrits dans la section V.2[2]. Dans la Figure V.16, nous avons le diagramme de structures composites détaillant la structure interne de *LEICASystem*. Remarquons que, dans ce diagramme, *LEICASystem*

[1] Les attributs et méthodes ont été cachés pour simplifier l'illustration.
[2] La classe *EvtNotificationSystem* ne représente pas vraiment un composant centralisé du système (il s'agit en fait d'un composant reparti). Elle a été définie ainsi à des fins de simulation.

représente un composant qui interagit avec son environnement (les acteurs/utilisateurs du système) par l'intermédiaire de ports.

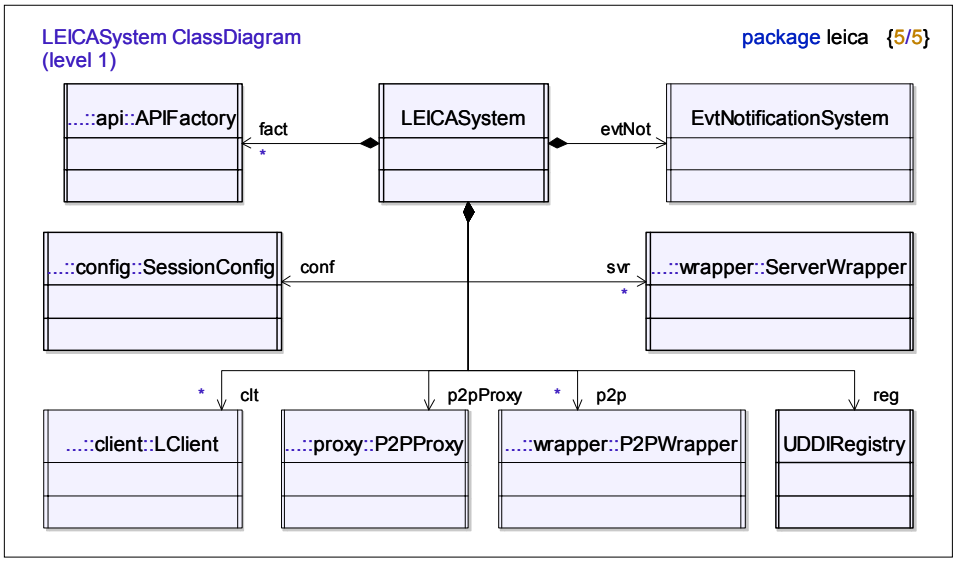

Figure V.15 Diagramme de classes de LEICA

Un port se comporte comme un point d'interaction avec l'environnement du composant. Un port est typé comme une interface, c'est-à-dire qu'il peut être fourni ou requis. Un port requis signifie qu'une instance du composant doit se connecter à une instance de composant fournissant le service demandé via un port fourni. Les interfaces de chaque port définissent l'ensemble des messages (ou signaux) que ce port est capable de recevoir ou d'envoyer. Un connecteur est une entité qui relie les ports des instances de composants. Ainsi, toujours dans la Figure V.16, les interactions entre les sous-composants de *LEICASystem* sont déterminées par des connecteurs liant leurs ports.

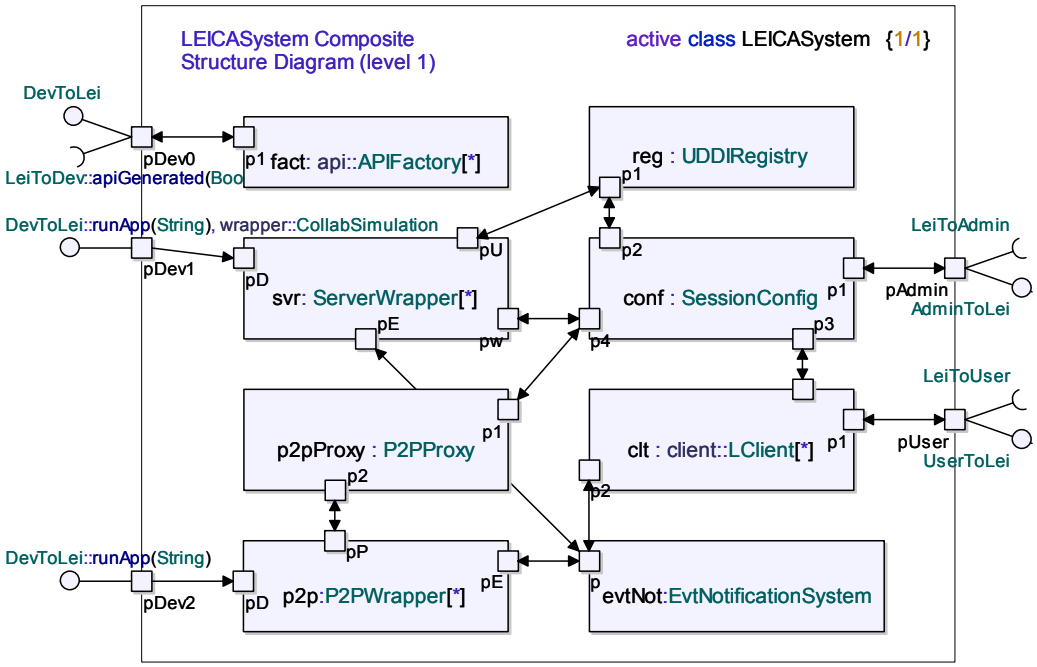

Figure V.16 Diagramme de structures composites de LEICASystem

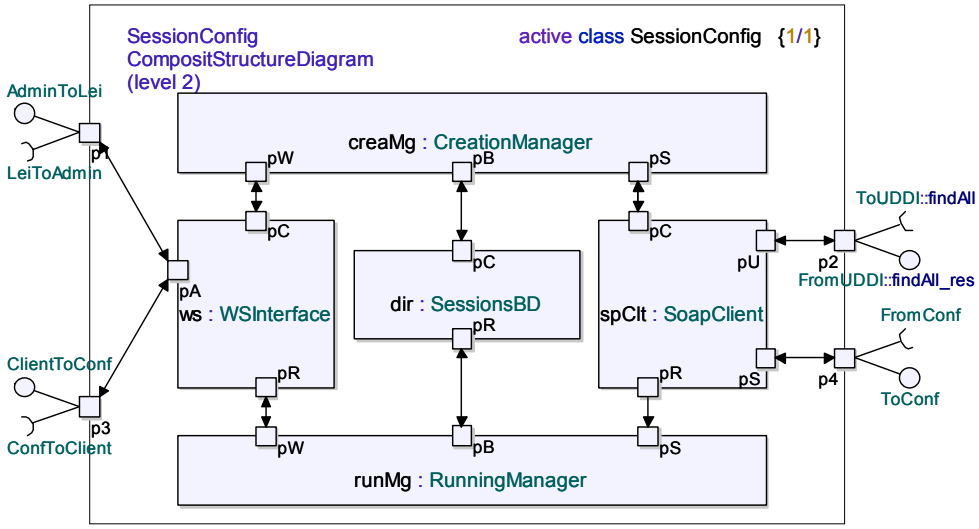

Figure V.17 Diagramme de structures composites du Session Configuration Service

Les deux diagrammes précédents ont été raffinés après la décomposition hiérarchique de l'architecture. Pour chaque composant nous avons défini des diagrammes de classes et de structures composites où nous avons spécifié leurs sous-composants. A titre d'exemple, la Figure V.17 montre le détail de la structure composite du *Session Configuration Service* et la Figure V.18 illustre la structure composite du *Server Wrapper*.

Au dernier niveau de raffinement (niveau 3), nous n'avons spécifié que des diagrammes de classes. Ce choix s'explique par le fait, qu'à l'intérieur de chaque sous-composant, nous n'avons pas été en mesure d'identifier des éléments qui puissent se comporter en tant que composants (c'est-à-dire, éléments indépendants communiquant par ports). Dans la Figure V.19 nous avons par exemple le diagramme de classes du *sous-composant SessionManager* du *ServerWraper* (présent dans le diagramme de la Figure V.18).

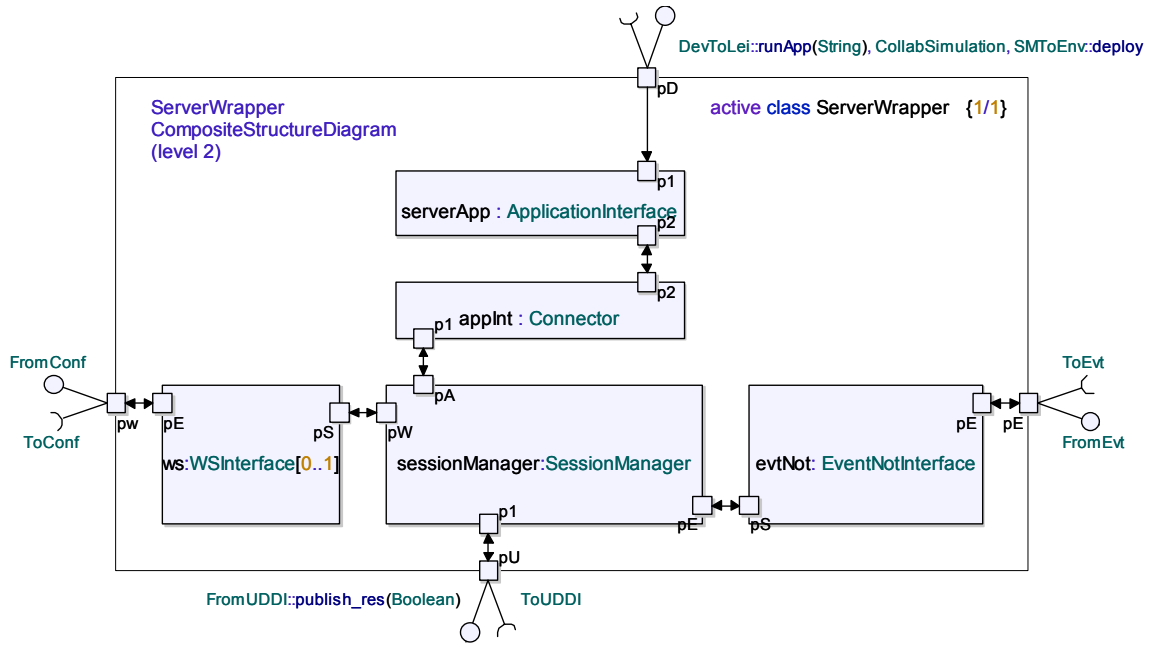

Figure V.18 Diagramme de structures composites du Server Wrapper

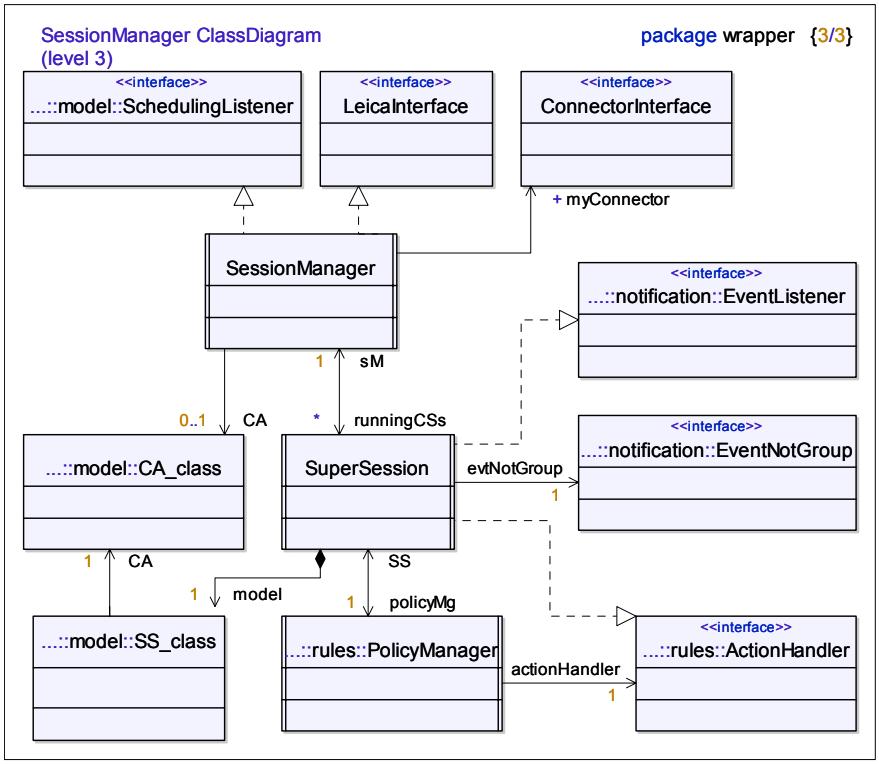

Figure V.19 Diagramme de classes du SessionManager du Server Wrapper

V.3.3.3. Comparaison des scénarios engendrés par *Tau G2* à ceux initialement proposés dans le cas des applications client/serveur

En nous appuyant sur l'utilisation du profil UML/SDL de l'outil *Tau G2*, nous avons décrit le comportement interne des différentes classes de notre système. Nous n'avons spécifié des diagrammes d'états que pour les composants devant être implantés lors du premier prototype de LEICA. Vu que dans ce prototype, nous avons décidé d'intégrer uniquement des applications client/serveur, nous n'avons pas pris en compte le *P2P Proxy*, le *P2P Wrapper*, et la partie du *Server Wrapper* concernant son comportement pour des applications multiserveurs. Dans cette première version, le comportement de l'application Web d'accès au *Session Configuration Service* n'a également pas été modélisé.

A titre d'exemple, la Figure V.20 illustre le diagramme d'états défini pour le sous-composant *CreationManager* du *Session Configuration Service*. En plus des diagrammes d'états nous avons défini les comportements de certaines méthodes de quelques classes par le biais de diagrammes textuels (qui correspondent en fait à du "pseudo-code"). C'est le cas de la Figure V.21 qui illustre la définition du comportement d'une méthode du sous-composant *SoapClient* du *Session Configuration Service*.

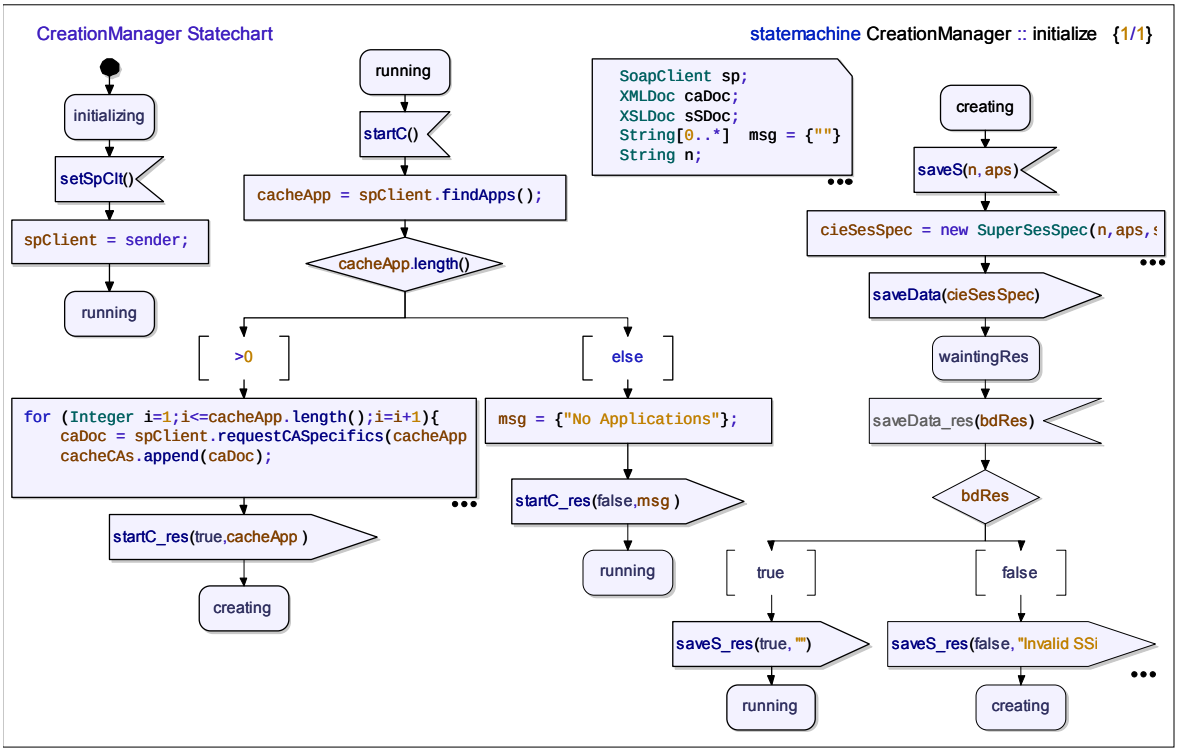

Figure V.20 Diagramme d'états du CreationManger, sous-module du Session Configuration Service

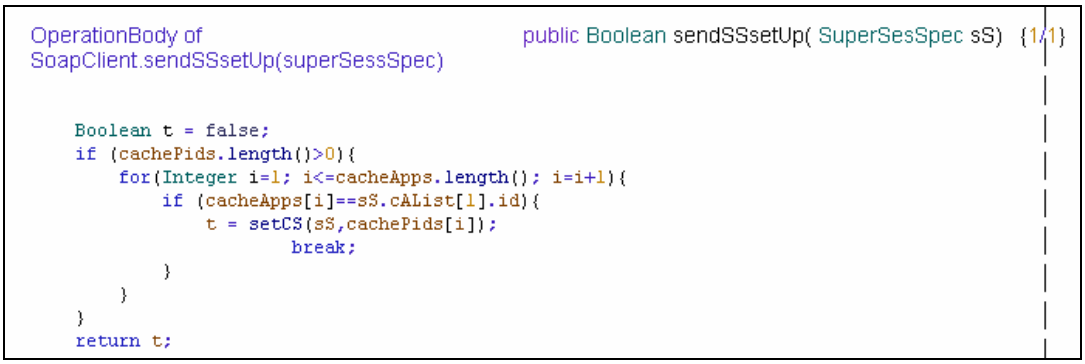

Figure V.21 Example de pseudo-code défini pour la méthode sendSSsetUp() de SoapClient, sous-composant du Session Configuration Service

À partir de la définition de ces comportements et en utilisant le module de simulation de l'outil *Tau G2*, nous avons exécuté diverses simulations en vue de valider l'architecture de notre système. Nous avons ainsi engendré des diagrammes de séquences qui ont été comparés aux diagrammes de séquences correspondants aux scénarios initialement proposés dans la phase d'analyse de la spécification de LEICA.

Les simulations ont été tout d'abord exécutées de manière individuelle pour chaque composant. Nous avons ainsi observé le comportement interne de chaque composant lorsque nous lui envoyons des messages en simulant son interaction avec l'environnement. L'environnement du point de vue d'un composant peut représenter soit un utilisateur soit d'autres composants. Suite à différentes simulations et aux comparaisons réalisées avec les scénarios définis préalablement, nous avons retouché plusieurs fois le comportement de chaque composant jusqu'à ce qu'il soit conforme à la spécification des interactions (diagrammes de séquences) définies initialement. Dans l'exemple de la Figure V.22 nous

présentons le diagramme de séquence engendré lors d'une simulation exécutée sur le composant *Server Wrapper*[1]. Dans cet exemple nous avons simulé une interaction avec le *Session Configuration Service* où ce dernier, lors du premier contact avec le *Server Wrapper*, lui demande des informations qui seront nécessaires pour la création d'une nouvelle *SuperSession*.

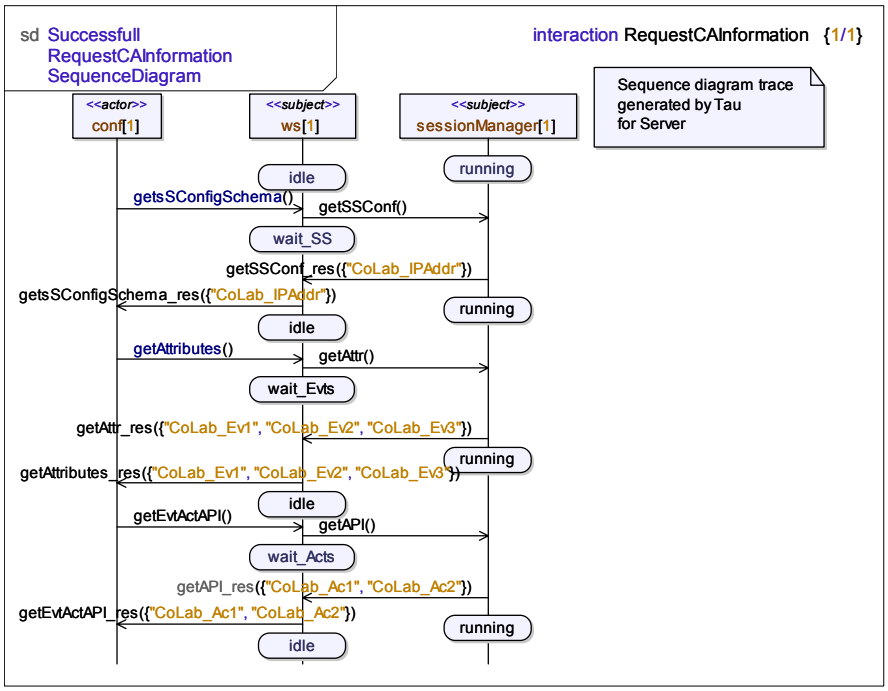

Figure V.22 Simulation d'une interaction entre le Server Wrapper (le composant testé) et le Session Configuration Service

Dans la suite, nous avons exécuté des simulations sur l'environnement de façon intégrale, en suivant les différents cas d'utilisations prédéfinis. Ainsi, des interactions avec les utilisateurs ont été simulées et des traces engendrées pour différentes situations, de succès ou d'erreur. Dans la Figure V.23, nous montrons le résultat d'une simulation concernant la création d'une *SuperSession*. Dans cette simulation, deux applications client/serveur ont été intégrées à LEICA[2]. Ce diagramme de séquences correspond en fait à une version plus détaillée de celui de la Figure V.12.

[1] Nous avons enlevé les sous-composants ne faisant pas partie de l'interaction.
[2] Nous avons supprimé du diagramme les sous-composants des *Server Wrappers* qui ne sont pas concernés par cette interaction.

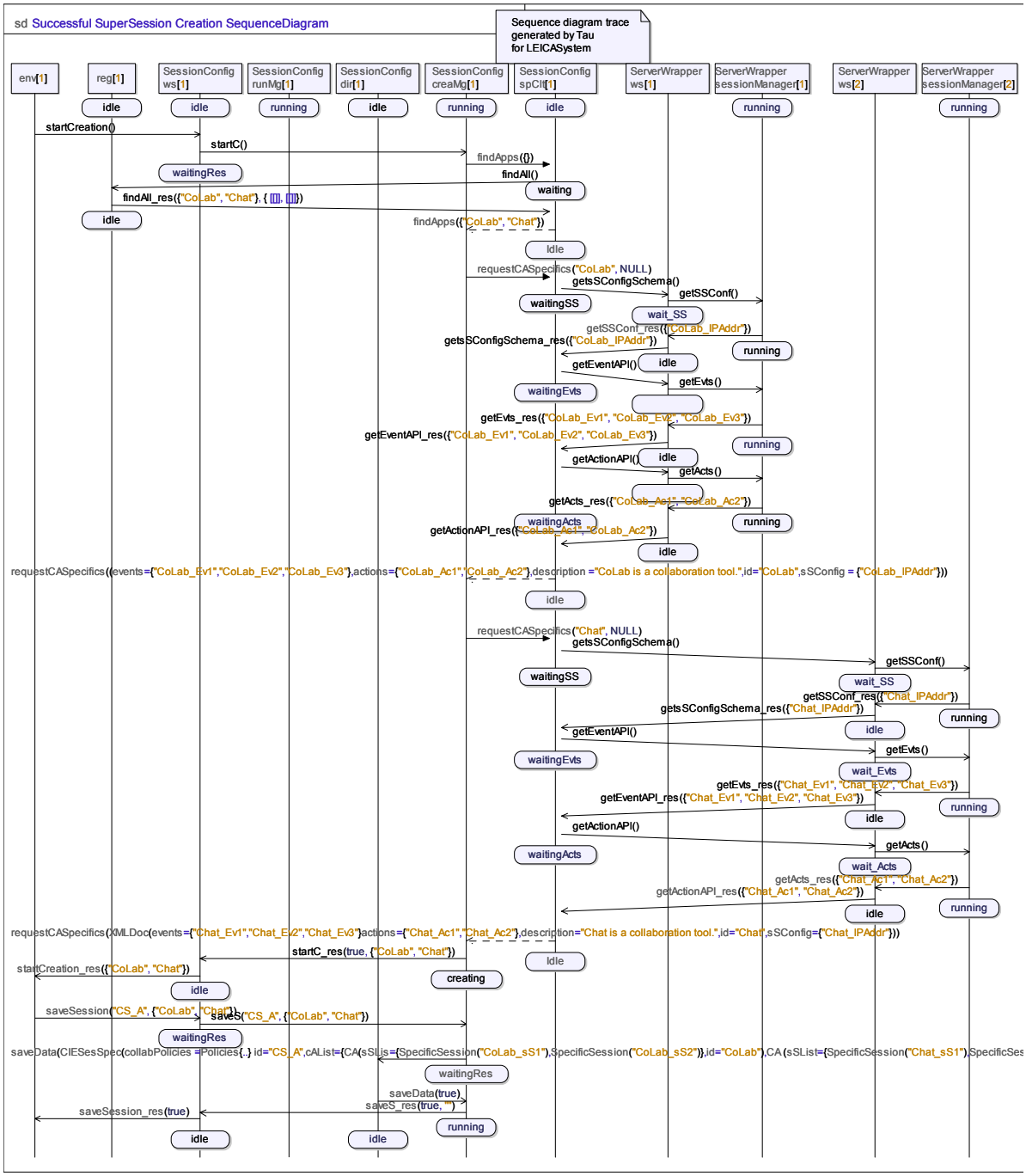

Figure V.23 Simulation de la création d'une SuperSession

V.4. Conclusion

Dans ce chapitre nous avons décrit l'architecture de LEICA en détaillant la dynamique de l'environnent et le fonctionnement de chaque composant pendant l'exécution de différentes activités, depuis l'intégration de nouvelles applications jusqu'à l'exécution de *SuperSessions*. Nous avons en particulier décrit le processus d'exécution répartie de la politique de collaboration d'une *SuperSession* et abordé les incohérences potentielles conséquences d'une telle approche repartie.

Nous avons ensuite introduit la méthode de modélisation utilisée pour la conception de notre environnement d'intégration. Cette méthode s'appuie sur le profil UML/SDL supporté par l'outil *Tau G2* de *Telelogic*. Nous avons validé la partie de l'architecture qui sera implémentée dans notre premier prototype au moyen de plusieurs simulations.

La modélisation de l'architecture selon le profil UML/SDL s'est réalisée dans une période d'environ 6 mois, le premier mois étant dédié à l'apprentissage de l'outil. Cette modélisation a été le point de départ de l'implémentation et du déploiement de LEICA, et elle s'est avérée fondamentale à ce stade d'implémentation, à cause de la complexité de la structure de LEICA.

Dans le chapitre suivant, nous allons décrire en détail l'implémentation du prototype actuel de LEICA. Notons que cette implémentation a été développée sans chercher à utiliser les facilités de génération automatique de code Java de l'outil *Tau G2*. Ceci est dû au fait que, à l'époque où nous avons initié ce développement, *Tau G2* ne disposait pas de générateur Java "complet" dans le sens où les comportements des classes n'étaient pas traduits (seuls les squelettes de l'architecture des classes l'étaient).

Chapitre VI

Implémentation et déploiement de LEICA

VI.1. Introduction

Nous allons présenter en détail différents aspects techniques concernant l'implémentation de LEICA. Dans la version actuelle de l'environnement, nous n'avons pas implémenté toutes les fonctionnalités qui ont été définies, mais un sous-ensemble suffisamment représentatif pour valider l'approche d'intégration proposée. Comme nous l'avons précisé dans le chapitre précédent, notre prototype présente deux limitations majeures par rapport à l'architecture de LEICA qui a été présentée auparavant :

(i) il ne supporte que l'intégration d'applications collaboratives client/serveur ;

(ii) l'application Web permettant d'accéder au *Session Configuration Service* n'a pas été implémentée, mais nous avons développé une application de test pour simuler le comportement de cette application Web.

Ce chapitre est organisé de la manière suivante : dans un premier paragraphe, nous détaillons les choix technologiques relatifs à l'implémentation de LEICA ; dans un deuxième paragraphe, nous présentons les principaux modules implémentés ainsi que quelques détails sur leur implémentation ; dans un troisième paragraphe, nous abordons l'intégration de deux applications collaboratives à l'environnement et décrivons la *SuperSession* créée dans le but de tester cette intégration.

VI.2. Les choix technologiques

Les critères qui nous ont guidé dans le choix des technologies adéquates pour la mise en œuvre de l'environnement ont été les suivants :

- la portabilité,
- la modularité,
- la simplicité de mise en oeuvre,
- l'utilisation de logiciels libres.

Pour offrir une solution portable, l'utilisation de Java [Java] s'est bien sûr imposée avec plusieurs plates-formes Java possibles[1]. Comme langage orienté objet offrant des mécanismes d'encapsulation, Java offre aussi la modularité. Le respect de ces critères passe également par l'utilisation du langage XML [XML], qui apporte également des éléments de réponse pour la modularité et la portabilité.

[1] Nous avons utilisé la version 1.4.2 du langage pour le développement.

Compte tenu de ces deux technologies de base, nous sommes allés chercher dans le domaine des logiciels libres des solutions pour mettre en œuvre les deux paradigmes de communication à déployer dans LEICA, à savoir les services Web et le système de notification d'événements basé sur le *publish*/*subscribe*.

VI.2.1. Les services Web

Pour qu'un service Web soit accessible, il doit être déployé dans un serveur d'application. Différentes plateformes étaient disponibles lors du développement de LEICA, mais les critères de simplicité et d'utilisation des logiciels libres nous ont guidé dans le choix du couple *Apache Tomcat 5.0* [*Tomcat*] et *Apache SOAP 2.3.1* [ApacheSOAP].

Tomcat est une implémentation *open source* des *servlets* Java développée dans le cadre du projet *Jakarta* de l'*Apache Software Foundation* [Apache]. Quant à *Apache SOAP*, il se compose d'un *servlet*, qui écoute les requêtes SOAP[1], et des classes Java nécessaires (i) pour traduire chaque requête en un appel du code de traitement du service Web respectif et (ii) pour traduire le résultat de cet appel en une réponse SOAP à envoyer au client. Dans LEICA, le code de traitement du service Web correspond aux classes Java implémentant le sous-module *WSInterface*, présent dans les *Wrappers* ainsi que dans le *Session Configuration Service*. La Figure VI.1 illustre le schéma d'interaction lorsque le *Servlet Apache SOAP* reçoit une requête par l'intermédiaire de *Tomcat*.

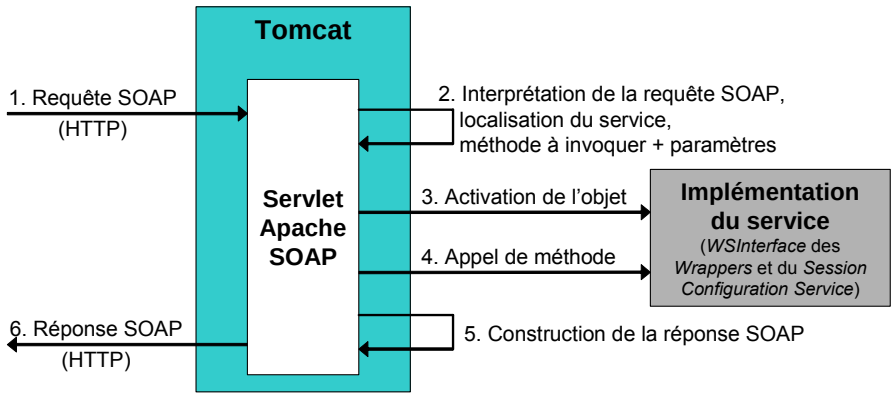

Figure VI.1 Le schéma d'interaction lors de la réception d'un message SOAP

Nous avons également utilisé l'API fournie par *Apache SOAP* pour le développement des modules clients qui doivent interagir avec les services Web. Pour constituer un environnement de développement et d'exploitation robuste, les services Web sont en général décrits par des méta-données exprimées en WSDL (*Web Service Definition Language*). Elles décrivent notamment les contenus des messages que le service prend en charge, et les modèles d'échange de messages valides pour un service (c'est-à-dire, le protocole de communication permettant de communiquer avec ce service). Un fichier WSDL décrit ainsi tous les éléments de base nécessaires à l'écriture d'un programme client capable d'interagir avec un service Web. Une limitation d'*Apache SOAP* est qu'il ne supporte pas la gestion de WSDL (au contraire de son successeur *Apache Axis* [Axis]).

Dans le cas de LEICA, étant donné que les clients SOAP qui peuvent interagir avec les interfaces de services Web de l'environnement (*i.e.* celles du *Session Configuration*

[1] Dans le cas de LEICA, les requêtes SOAP sont enveloppées dans des requêtes HTTP.

Service et des *Wrappers*) connaissent à priori la définition de ces interfaces, nous pouvons facilement nous passer de WSDL. Ainsi, nos clients SOAP utilisent des *stubs* (ou souches) statiques, c'est-à-dire, des objets locaux implémentés préalablement et qui se comportent comme des *proxies* des services Web respectifs.

Rappelons cependant que LEICA prévoit le déploiement dynamique d'interfaces de services Web dans le cas des *Wrappers*, au fur et à mesure que des nouvelles applications sont intégrées à l'environnement. Ainsi, même si le *Session Configuration Service* (ou plus précisément, son client SOAP) connaît la définition des interfaces des *Wrappers*, il doit également connaître leur emplacement. C'est ainsi que le *UDDI Registry* privé joue un rôle important dans notre environnement.

UDDI (*Universal Description, Discovery, and Integration*) définit un protocole d'interrogation et de mise à jour d'un annuaire. Un registre UDDI représente donc un annuaire qui contient des informations détaillées concernant les services Web et leurs emplacements. Une entrée d'annuaire UDDI se compose de trois parties principales, à savoir le fournisseur du service, les services Web offerts et les liens vers les implémentations. Dans le cas de LEICA, cette dernière partie est la plus importante puisqu'elle associe l'entrée de service Web à l'URI exact identifiant l'emplacement du service. Ainsi, chaque *Wrapper* implémente un client UDDI capable de publier cette information dans le *UDDI Registry* privé. De même, le *Session Configuration Service* possède un client UDDI pour faire des recherches dans le *UDDI Registry* privé.

Pour implémenter les clients UDDI, nous avons utilisé *UDDI4J v.2* [UDDI4J]. Originalement rendu disponible par IBM, *UDDI4J* est un projet *open source* qui présente une bibliothèque Java mettant à disposition une API pour interagir avec les annuaires UDDI. Quant à l'*UDDI Registry* privé, nous avons utilisé *jUDDI* [jUDDI], une implémentation *open source* qui peut être considérée comme le serveur UDDI officiel de la fondation *Apache*. Associée à *jUDDI*, nous avons utilisé *MySQL* [MySQL] en tant que base de données. *jUDDI* fournit également une API de développement pour créer des clients UDDI. Nous avons cependant choisi d'implémenter les clients UDDI en utilisant *UDDI4J* à cause de sa simplicité, mais aussi dans le but de vérifier l'interopérabilité entre ces deux APIs.

VI.2.2. Le paradigme *publish*/*subscribe*

Le paradigme *publish*/*subscribe* peut être mis en œuvre selon deux politiques principales d'abonnement (ou souscription) : l'abonnement basé sur le sujet de l'événement, (de l'anglais *subject-based* ou *topic-based*), et l'abonnement basé sur le contenu (de l'anglais *content-based*).

Les systèmes de notification d'événements par abonnement les plus répandus s'appuient sur le sujet de l'événement. Dans ces systèmes, chaque événement est classifié comme appartenant à un type de sujet prédéfini. Les producteurs sont donc tenus d'associer à leurs événements un des sujets prédéfinis, et les souscripteurs doivent s'abonner à l'un de ces sujets prédéfinis.

Une alternative est la technique s'appuyant sur le contenu. Ce type d'abonnement apporte une flexibilité supplémentaire pour les abonnés. Le choix du filtrage parmi les différentes valeurs des attributs des événements permet d'éviter de prédéfinir les sujets. Cette approche dynamique exige cependant des protocoles beaucoup plus complexes, notamment pour ce qui concerne les mises en correspondance (*matching*) entre les

souscriptions et les publications. Cela implique donc une surcharge considérable lors de l'exécution du système[1].

Dans le cas de LEICA, nous avons été encouragés à employer la politique d'abonnement s'appuyant sur le sujet, étant donné que nous connaissons préalablement tous les types d'événements qui peuvent être notifiés lors de l'exécution d'une *SuperSession*. Nous avons trouvé différentes technologies disponibles en logiciel libre pour implémenter cette approche. Notamment des implémentations de la spécification JMS [JMS] – une API Java offrant une interface d'accès à des services de messagerie qui supporte le paradigme *publish/subscribe*. La plupart de ces implémentations utilisent l'approche s'appuyant sur des *Message Brokers* (ou "courtiers de message"), qui représente un composant logiciel gérant l'échange de messages entre les applications (par exemple [Narrada] et [Proteus]).

Les *Wrappers* de LEICA doivent échanger entre eux des notifications d'événement en tant que *peers* (ou pairs), c'est-à-dire, sans avoir à passer par des médiateurs, écartant ainsi les problèmes liés au passage à l'échelle. Des infrastructures telles que *Joram* [Joram], *MantaRay* [MantaRay], *Scribe* [Scribe] proposent des solutions pour une mise en œuvre complètement distribuée sur une architecture pair à pair. L'inconvénient majeur de *Joram* et *MantaRay* est l'absence d'un mécanisme de diffusion de messages par *multicast*, qui pourrait optimiser l'envoi lors que différents souscripteurs sont associés à un même type de sujet d'événement.

Ainsi, nous avons choisi *Scribe* pour mettre en œuvre le système de notification d'événements de LEICA. *Scribe* est un système écrit en Java qui implémente le paradigme *publish/subscribe* basé sur des sujets. Il est construit au dessus de *Pastry*[2] [Pastry], un substrat générique et qui passe à l'échelle pour la communication pair à pair.

VI.2.2.1. *Pastry* et *Scribe*

Dans *Pastry*, l'algorithme de routage des messages s'appuie sur les notions de clés numériques et de *nodeIds*. Une clé numérique est un nombre qui peut être employé pour identifier un objet applicatif ; cette clé est choisie dans un espace d'identificateurs assez grand. Chaque nœud *Pastry* du réseau logique porte un identificateur numérique, ou *nodeId*, choisi aléatoirement avec une probabilité uniforme dans cet espace. *Pastry* assigne chaque clé d'objet au nœud possédant le *nodeId* numériquement le plus proche de la clé. Ce nœud devient ainsi le responsable de la clé. Lorsqu'un nœud reçoit un message portant une clé numérique, il réalise, de manière efficace, le routage du message vers le nœud de *nodeId* le plus proche numériquement de la clé (parmi tous les nœud actifs du réseau logique).

Le réseau logique de nœuds *Pastry* est auto-organisable et auto-réparable ; chaque nœud maintient une table de routage comprenant O(log(*n*)) entrées (*n* est le nombre de nœuds du réseau logique). Chaque entrée fait l'association entre un *nodeId* et son adresse IP. Un message peut être routé vers le nœud responsable d'une clé donnée en O(log(n)) sauts. La proximité topologique des nœuds est exploitée lors de la construction du réseau logique.

---

[1] Il existe également une solution légèrement différente du critère *content-based*, mais avec les mêmes inconvénients de complexité d'implémentation, qui s'appuie sur des modèles d'événement (*pattern-based*). Dans ce cas, on ne se préoccupe plus des valeurs des différents attributs des événements, mais plutôt du type de ces attributs.

[2] La version *open source* s'appelle *FreePastry* et elle a été développée par l'université RICE à Houston.

Pastry est ainsi un système complètement décentralisé du fait que toutes les décisions s'appuient sur des informations locales à un nœud et tous les nœuds ont les mêmes capacités. Néanmoins, *Pastry* définit un nœud unique de *bootstrap*, qui peut être vu comme un "point de départ" pour de nouveaux nœuds désirant rejoindre le réseau logique. Mais n'importe quel nœud du réseau peut jouer ce rôle de *bootstrap*.

En s'appuyant sur *Pastry*, *Scribe* implémente le paradigme *publish/subscribe* par le biais de groupes multicast. Chaque groupe correspond à un sujet[1] et est associé à une clé appelée *groupId*. Pour créer un group, *Scribe* demande à *Pastry* de router un message *CREATE* un utilisant le *groupId* comme clé. Le nœud *Pastry* responsable de cette clé devient alors la racine de l'arbre multicast correspondant à ce groupe. Il est désigné en tant que point de rendez-vous (*rendez-vous point*) associé au groupe et utilisé pour disséminer des événements publiés sous ce sujet.

Pour joindre un groupe et devenir un souscripteur, *Scribe* demande à *Pastry* d'envoyer un message *JOIN* portant le *groupId* correspondant comme clé. Ce message est dirigé vers le point de rendez-vous du groupe par *Pastry*. En employant un schéma similaire au *Reverse Path Forwarding* (RPF) [Dalal-78] l'arbre multicast est formé en fusionnant les routes qui partent de tous les souscripteurs jusqu'à sa racine. Le mécanisme de génération de l'arbre multicast est ainsi complètement décentralisé. Ensuite, les nœuds peuvent potentiellement joindre et quitter différents groupes plusieurs fois, ce qui amortit le coût initial associé à la construction du réseau logique de *Pastry*.

Pour publier un événement associé à un sujet, *Scribe* demande à *Pastry* d'envoyer un message *PUBLISH*, avec le *groupId* correspondant. Le message arrive au point de rendez-vous du groupe et *Pastry* se charge ensuite de le disséminer à travers son arbre multicast.

Scribe utilise TCP pour disséminer les messages de façon fiable en même temps qu'il utilise *Pastry* pour réparer l'arbre multicast dans le cas où un de ses nœuds est défaillant. Par défaut, une livraison fiable et ordonnée des messages ne peut être garantie que si les connexions TCP entre les nœuds ne tombent pas. En cas de problèmes au niveau de TCP, nous aurions cependant une livraison ordonnée des événements en provenance d'un même producteur. *Scribe* prévoit des mécanismes simples pour permettre aux applications de mettre en œuvre des garanties plus fortes de fiabilité et d'ordonnancement.

VI.3. Principaux modules du prototype

Les quatre modules de LEICA implémentés dans notre prototype sont : l'*APIFactory*, le *Server Wrapper*, le *Session Configuration Service* et le *LClient*. Le tableau, illustré dans la Figure VI.2, montre les différents paquetages Java créés lors de l'implémentation ainsi que leur utilisation par chacun de ces modules. Nous avons développé au total 79 classes Java (9500 lignes de code, écrites pendant une période de 7 mois de développement) dans le but d'implémenter ces modules. Nous allons présenter maintenant quelques détails concernant l'implémentation des modules constituant le prototype de LEICA.

[1] Dans *Scribe*, le terme utilisé est *topic*.

Nom du paquetage	Description	Utilisation			
		API Factory	*Server Wrapper*	*Session Configuration Service*	*LClient*
leica.wrapper.api	Des classes spécifiques au *APIFactory*.	X			
leica.wrapper	Des classes spécifiques au *Server Wrapper*.	X	X		
leica.config	Des classes spécifiques au *Session Configuration Service*.			X	
leica.client	Des classes spécifiques au *LClient*.				X
leica.xml	Des classes utilitaires pour le traitement de données XML.	X	X	X	X
leica.model	Des classes correspondant au modèle de la *SuperSession*, utilisées dans la gestion de l'état global de la session.		X		X
leica.data	Définition des types de messages échangés pendant l'exécution d'une *SuperSession*.		X		X
leica.notification	Des classes pour la mise en œuvre du système de notification d'événements.		X		X
leica.rules	Des classes qui correspondent au sous-module d'exécution des politiques de collaboration.		X		X

Figure VI.2 Les paquetages Java implémentés dans le prototype de LEICA

VI.3.1. *APIFactory*

La classe *APIFactory* constitue une application Java qui doit recevoir comme paramètres les deux fichiers décrivant l'application à être intégrée : le fichier de données spécifiques et le fichier d'attributs et d'API. Le premier fichier doit être spécifié en tant qu'un *XML Schema* [XMLSchema] et le deuxième en tant qu'un fichier XML simple. Nous avons choisi le format *XML Schema* pour le fichier de données spécifiques dans le but de faciliter le développement futur de l'application Web qui sera utilisée pour configurer des *SuperSessions*[1]. Dans l'Annexe A, nous présentons et expliquons le format *XML Schema* défini pour la spécification de fichiers de données spécifiques, ainsi que le format XML des fichiers d'attributs et d'API.

La classe *APIFactory*, ainsi que toutes les autres classes utilisées dans la création dynamique d'un *Wrapper* se trouvent rassemblées dans le ficher "leica.jar"[2]. Pour engendrer un *Wrapper*, il suffit d'exécuter la commande :

```
prompt> java –jar leica.jar fichier1.xml fichier2.xsd
```

[1] L'utilisation de *XML Schema* nous permettra de générer automatiquement les interfaces utilisateur pour la configuration de l'*IAInfo*.

[2] Un fichier ".jar", ou *Java ARchive*, est un fichier d'archive compressé réunissant des classes Java, ainsi que d'autres types de fichiers.

et ensuite, la classe *APIFactory* se charge d'interpréter les fichiers passés en paramètres et de créer le sous-module *Application Interface* pour le *Wrapper*. Cela consiste à créer et compiler, en temps d'exécution de nouvelles classes/interfaces Java qui sont des spécialisations de quelques interfaces Java prédéfinies. La génération dynamique du code s'appuie sur le principe d'introspection. L'introspection est une technique que l'on retrouve dans quelques langages permettant à un programme de regarder à l'intérieur d'une classe dans le but de découvrir dynamiquement des informations concernant cette classe. Ce mécanisme permet à un environnement donné d'intégrer dynamiquement de nouveaux types qui ne sont pas connus au préalable [1].

Dans la Figure VI.3, nous avons le diagramme de classes illustrant les classes et interfaces créées et/ou utilisées lors de la création dynamique du sous-module *Application Interface*.

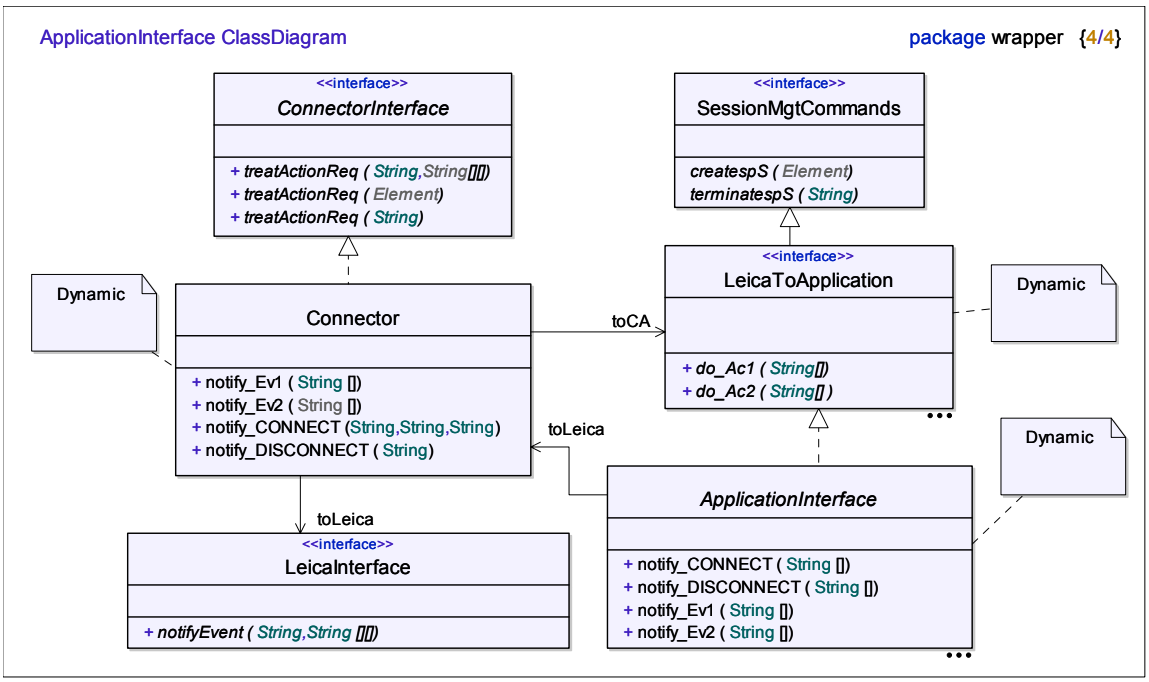

Figure VI.3 Les classes et interfaces Java créées dynamiquement pendant la création d'un Wrapper

Une fois que le *APIFactory* a compilé ces nouvelles classes, il engendre deux nouveaux fichiers .jar : "Wrapper.jar" et "ws.jar". Le premier fichier contient toutes les classes du *Server Wrapper*, à l'exception de celles qui composent le sous-module *WSInterface*, qui sont présentes dans le deuxième fichier.

Le fichier "Wrapper.jar" représente une bibliothèque Java qui doit être utilisée lors de l'intégration du *Wrapper* à l'application. Comme nous l'avons dit dans la section V.2.1.1, nous devons, pour accomplir cette intégration, coupler le sous-module *Application Interface* au serveur de l'application collaborative. Notons dans la Figure VI.3 que la classe *ApplicationInterface*, qui correspond à ce sous-module, est définie en tant que classe abstraite [2]. Elle contient une liste de méthodes abstraites [1] : des méthodes du type

[1] La *Reflection API* [Green-98] de Java s'appuie sur la réflexivité (de l'anglais *reflection*) en tant que moyen pour aboutir à l'introspection. Les classes nécessaires à l'introspection se trouvent dans les paquetages *java.lang* et *java.lang.reflect*.

[2] En UML les noms de classes et méthodes abstraites apparaissent en italique.

do_ActionType() (héritées de *LeicaToApplication*), les méthodes *createspS()* et *terminatespS()* (héritées de *SessionMgtCommands*).

Ainsi, lors de l'intégration du *Wrapper* à l'application, le développeur doit créer une nouvelle classe qui hérite de *ApplicationInterface* en implémentant ses méthodes abstraites – dans le codage de ces méthodes, des appels aux fonctions de l'API de l'application doivent être réalisés afin de traiter les requêtes d'établissement (et d'arrêt) de sessions spécifiques, ainsi que des requêtes d'exécution d'actions. Ensuite, le développeur doit compléter l'implémentation en associant les méthodes *notify_EventType()* (implémentées dans la classe *ApplicationInterface*) aux fonctions de l'API de l'application permettant à cette dernière de notifier l'occurrence d'événements dans son activité collaborative. Le fait que le *Wrapper* soit implémenté en Java n'empêche pas que des applications implémentées dans d'autres langages soient intégrées à LEICA. Nous pouvons toujours coupler du code Java à des applications développées dans d'autres langages par le biais de l'API JNI [2].

En ce qui concerne le ficher "ws.jar", nous avons décidé de mettre les classes qui implémentent le sous-module *WSInterface* (notamment les classes présentes dans le paquetage Java *leica.wrapper.ws*) dans ce deuxième fichier d'archivage pour simplifier le processus de déploiement des services Web. Il suffit ainsi de copier ce fichier dans un répertoire spécifique de *Tomcat* défini selon la configuration de ce dernier. Cela permettra à *Apache SOAP* de trouver les classes du *WSInterface* et d'invoquer les méthodes de ces classes lorsqu'il reçoit une requête SOAP dirigé au *Wrapper*.

VI.3.2. *Server Wrapper*

Une fois que le *Server Wrapper* a été couplé au serveur d'une application collaborative, il suffit d'exécuter ce serveur pour que le *Wrapper* soit également exécuté. Dans la Figure V.18 présentée dans le chapitre précédent, nous avons illustré les sous-modules du *Server Wrapper*. Initialement, juste les objets des classes *ApplicationInterface*, *Connector*, *EventNotInterface* et *SessionManager* sont activés. Le *WSInterface* n'est créé que lorsque le *servlet* SOAP reçoit une première requête dirigée au *Wrapper*.

Le *SessionManager* démarre son activité en déployant l'interface de services Web du *Server Wrapper* pour ensuite la publier dans le *UDDI Registry* privé de LEICA. Le déploiement correspond en fait à déclarer au *servlet* SOAP les méthodes de l'interface de telle façon qu'il puisse les référencer plus tard quand le service est appelé. En d'autres termes, le *servlet* SOAP ne peut invoquer que des méthodes de services Web qu'il connaît, et déployer un service est le moyen de les lui faire connaître. Ainsi, pendant le déploiement de l'interface de services Web, le *Server Wrapper* informe au *servlet* SOAP, entre autres données, l'identificateur de l'interface[3] et sa liste de méthodes. Nous trouvons dans cette liste les méthodes suivantes :

- *getAttributes()* – Méthode qui fournit en réponse un document XML contenant les attributs qui décrivent l'application collaborative.
- *getsSConfigSchema()* – Méthode qui fournit en réponse un document XML spécifiant les paramètres de configuration nécessaires à la création d'une session spécifique pour cette application collaborative.

[1] Méthodes qui ne contiennent pas d'implémentation, juste leur définition.
[2] *Java Native Interface* – interface de programmation permettant d'utiliser du code natif dans une classe Java.
[3] Les interfaces de services Web des *Wrappers* sont identifiées par "urn:leica:*applicationID*".

- *getEvtActAPI()* – Méthode qui fournit en réponse un document XML décrivant l'API événements/actions (et des particularités de l'API de gestion) de l'application.
- *setSession()* – Méthode permettant d'informer l'application collaborative qu'elle doit faire partie d'une *SuperSession* en cours de démarrage. Cette méthode reçoit en paramètre un document XML contenant les informations de configuration de la *SuperSession*[1] et fournit en réponse une valeur de type *Integer*. Si cette valeur est négative, ceci indique que l'application n'a pas réussi à exécuter les procédures nécessaires au démarrage de la *SuperSession*.
- *delSession()* – Méthode permettant d'informer l'application collaborative qu'une *SuperSession* s'est arrêtée. Cette méthode reçoit en paramètre le nom de la *SuperSession* et fournit en réponse un *boolean* indiquant si l'application a bien réussi à arrêter les composants associés à la *SuperSession*.

Ainsi, quand le *servlet* SOAP reçoit une requête invoquant une de ces méthodes, il active l'objet de la classe *WSInterface* pour qu'elle puisse traiter cette requête (et éventuellement des requêtes suivantes). Remarquons que le module *WSInterface* est exécuté par la machine virtuelle Java (JVM, de l'anglais *Java Virtual Machine*) associée au serveur *Tomcat*. Autrement dit, le *WSInterface* n'est pas exécutée par la même JVM utilisée pour lancer le serveur de l'application collaborative et son *Wrapper*. Par conséquent, il n'est pas capable de communiquer avec les autres modules du *Wrapper* (notamment le *SessionManager*) en faisant des appels de procédures locales. En fait, différentes solutions sont envisageables pour mettre en place la communication entre deux objets Java exécutés dans deux JVMs distinctes (localisées ou non dans une même machine) : (i) *JMS* [JMS] (communication par messages) ; (ii) *RMI* [Sun-03] (communication par invocation de méthodes distantes) ; (iii) *sockets TCP*. Nous avons décidé d'employer la dernière option (connexion de type *socket* sur *localhost*) puisqu'elle s'avère beaucoup plus simple et performante. RMI et JMS risquent fort de représenter des solutions trop complexes (pour une simple communication locale) introduisant des surcharges considérables [Laramee-02] [Morgan-97].

Lorsque le *Server Wrapper* (par l'intermédiaire de son *WSInterface*) reçoit un appel à la méthode *setSession()*, son *SessionManager* exécute les procédures nécessaires au démarrage de la *SuperSession*. Il crée un objet de la classe *SuperSession*, qui à son tour crée les sous-modules pour la gestion d'état et pour l'exécution de la politique de collaboration de la *SuperSession*, respectivement implémentées par les classes *SS_class* et *PolicyManager* (nous avons illustré ces classes dans la Figure V.19 présentée dans le chapitre précédent). Ensuite, le *SessionManager* demande au sous-module *Event Notification Interface* (implémenté par la classe *EvtNotificationSystem*) de se connecter au système de notification d'événements de cette *SuperSession*.

VI.3.2.1. La mise en place du système de notification d'événements

Lorsque la classe *EvtNotificationSystem* reçoit l'appel du *SessionManager* pour se connecter au système de notification d'événements d'une *SuperSession*, ce dernier lui repasse l'adresse du nœud de *bootstrap* (une adresse IP et un numéro de port TCP) pour qu'il puisse le contacter et ainsi rejoindre le réseau logique de *Pastry* associé à la *SuperSession*.

[1] Chaque *Wrapper* ne reçoit dans le *IAInfo* que les informations concernant l'application à laquelle il est associé.

Normalement, les *Wrappers* reçoivent les informations concernant le nœud de *bootstrap* dans le document XML de configuration de la *SuperSession* (reçu en tant que paramètre de la méthode *setSession()*). Néanmoins, le premier *Wrapper* à recevoir une requête *setSession()* pour une *SuperSession* donnée ne reçoit pas ces informations. Ceci indique que ce *Wrapper*, ou plus précisément son nœud *Pastry*, doit être le nœud de *bootstrap* pour cette *SuperSession*. Ainsi, le *EvtNotificationSystem* crée le composant *Scribe* (et son nœud *Pastry* associé) et renvoie en retour au *SessionManager* le numéro de port utilisé par *Pastry*. Remarquons qu'un même *Wrapper* peut être impliqué dans différentes *SuperSessions*. De ce fait, le *EvtNotificationSystem* doit créer pour chaque *SuperSession* un nouveau composant *Scribe*.

Finalement, en réponse à la requête *setSession()*, le *Wrapper* envoie au *Session Configuration Service* le numéro de port utilisé par le nœud de *bootstrap* de *Pastry*. Par la suite, le *Session Configuration Service* rajoute au document XML de configuration de la *SuperSession* l'adresse IP du premier *Wrapper* contacté et le numéro de port que ce dernier lui a envoyé. Le document modifié est envoyé aux autres *Wrappers* par des requêtes *setSession()* à leurs interfaces de services Web.

Rappelons que, dans la section V.2.3.5, nous avons mentionné qu'un des *Wrappers* utilisés lors de l'exécution d'une *SuperSession* est automatiquement désigné en tant que gestionnaire d'état (responsable de l'envoi à tout nouvel entrant d'une notification d'événement contenant l'état actuel de la *SuperSession*). Ainsi, nous avons choisi de toujours désigner le *Wrapper* créateur du nœud de *bootstrap* en tant que gestionnaire d'état d'une *SuperSession*.

Une fois le système de notification d'événements mis en place, chaque *Wrapper* peut publier des événements et s'abonner aux sujets associés aux types de ces événements. Par exemple, parmi l'ensemble des types d'événements auxquels le *EventNotInterface* devra s'abonner, nous avons ceux appartenant à l'API de gestion. Cela permet au *SS_class* d'être informé des changements concernant les composants actifs (dynamiques) de l'état de la *SuperSession*.

Puisque chaque application intégrée à LEICA peut définir ses propres types d'événement, cela risque de causer des conflits. Deux applications peuvent éventuellement choisir de donner le même nom à un type d'événement (bien que les types d'événement soient différents puisqu'ils ont été engendrés par des applications distinctes). Pour surmonter ce problème, nous avons défini que chaque sujet doit inclure l'identificateur de l'application (*CAid*) en plus du type d'événement. Ainsi chaque sujet est nommé par "CAid:type_d'événement". Dans le cas des événements de l'API de gestion relatifs à LEICA, les sujets sont nommés "LEICA:type_d'événement".

VI.3.2.2. Le *PolicyManager* et l'exécution de la politique de collaboration

Initialement, le *PolicyManager* interprète les données XML spécifiant la politique de collaboration de la *SuperSession* et met en place l'exécution des règles politiques. Il crée pour chaque règle à être exécutée dans ce *Wrapper* un objet de la classe *Rule*, illustrée dans la Figure VI.4. Cette classe implémente un *thread* Java qui reste en sommeil jusqu'à ce que des notifications d'événement lui soient communiquées. Lors d'une notification, le *thread* est réveillé pour exécuter le processus de sensibilisation de la règle selon la sémantique que nous avons définie dans la section IV.4.2.4. Dans la Figure VI.4 nous

illustrons toutes les classes (appartenant au paquetage *leica.rule* [1]) permettant d'implémenter la structure d'une règle selon sa spécification XML.

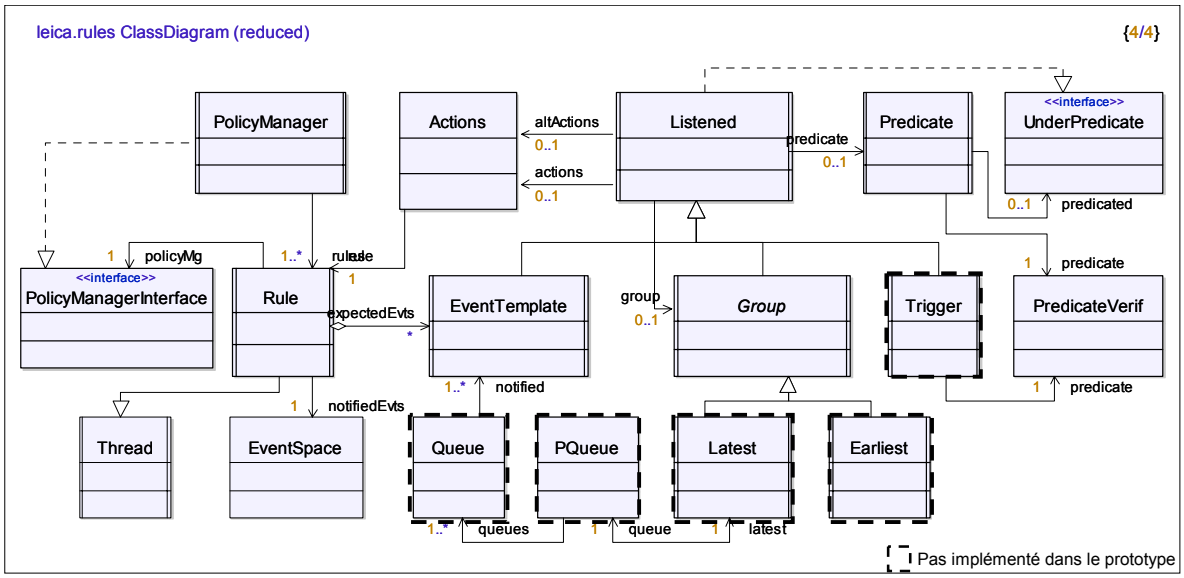

Figure VI.4 Diagramme de classes du paquetage leica.rules

Toutes les règles sont exécutées en parallèle et indépendamment les unes des autres. Une même notification d'événement peut ainsi être à l'origine de la sensibilisation de différentes règles.

Une particularité concernant l'exécution des règles est que nous permettons au créateur de la *SuperSession* de définir le "temps de vie" de chaque règle à travers le paramètre *ttl*. Si *ttl=n* (*n*>0), la règle pourra être sensibilisé (et donc tirée) *n* fois, devenant inactive après son *n*-ième tir (c'est-à-dire, elle ne pourra donc plus être exécutée). Si *ttl=**, la règle sera active (donc en exécution) pendant tout le temps d'exécution de la *SuperSession*, pouvant être tirée autant de fois que nécessaire. Ce dernier cas sera considéré comme comportement par défaut dans le cas où *ttl* n'est pas spécifié.

Pour l'instant, dans notre prototype opérationnel, nous ne supportons que des règles qui associent 1 événement[2] à *n* actions. Du fait que nous n'avons pas de règles utilisant des *Latest*, il suffit de rendre fiable notre système de notification d'événements (*Scribe*) pour garantir la cohérence lors de l'exécution des règles. De ce fait, pour le moment, nous n'avons pas implémenté l'ordonnancement des notifications d'événements.

VI.3.3. *Session Configuration Service*

Le *Session Configuration Service* est un composant qu'interagit avec le reste de l'environnement par le biais uniquement des services Web. Il reçoit des requêtes SOAP pour des appels aux méthodes de son interface de services Web à travers son sous-module *WSInterface*. Lorsqu'il traite certaines de ces méthodes, il peut également contacter d'autres services Web (le *UDDI Registry* et les *Wrappers*) en utilisant son sous-module *SoapClient*. Remarquons que les sous-modules du *Session Configuration Service* ont été illustrés dans la Figure V.17 du chapitre précédent.

[1] Quelques classes non relevantes à ce moment ont été retirées du diagramme.
[2] Nous supportons pour l'instant que des *Events*, les *Triggers* n'ont pas encore été implémentés.

L'interface de services Web du *Session Configuration Service* est déployée manuellement dans le *servlet* SOAP et ses méthodes peuvent être classées en trois groupes. Le premier groupe concerne les méthodes traitées par le sous-module *CreationManager* qui doivent être utilisées lors de la création d'une nouvelle *SuperSession*. Ces méthodes sont :

- *startCreation()* – Méthode permettant de signaler au *Session Configuration Service* le démarrage du processus de création et de configuration d'une *SuperSession*. Lorsque cette méthode est appelée, le *CreationManager* contacte le *UDDI Registry* privé (en utilisant son client UDDI imbriqué dans le sous-module *SoapClient*) pour avoir la liste actuelle des applications intégrées à LEICA. Il renvoie ensuite, en réponse, cette liste (le nom de chaque application plus une petite description). Après avoir répondu à la méthode *startCreation()*, le *CreationManager* déclenche en tant que tache de fond un processus en charge de contacter les applications présentes dans cette liste. Pour chacune de ces applications, il appelle[1] les méthodes *getAttributes()*, *getsSConfigSchema()* et *getEvtActAPI()* (définies dans l'interface de services Web de chaque application). Les informations obtenues sont enregistrées dans un *cache* local.
- *getAttr()*, *getAPI()*, *getsSspec()* – Méthodes qui reçoivent en paramètre l'identificateur d'une application et permettent de demander au *Session Configuration Service* les informations concernant (i) les attributs, (ii) l'API événements/actions, et (ii) la liste de paramètres spécifiques (nécessaires à la création de sessions spécifiques) de cette application. Le *CreationManager* récupère ces informations de son *cache* local et les renvoie en réponse. Si les informations ne sont pas encore disponibles dans le *cache*, il contacte l'application pour lui demander ces informations.
- *saveSession()* – Méthode qui reçoit en paramètre un document XML contenant les données de configuration d'une *SuperSession*. Tout d'abord, le *CreationManager* s'assure que l'identificateur de la *SuperSession* est bien unique. Si c'est le cas, il crée un fichier[2] pour sauvegarder localement les informations de configuration. Finalement le *CreationManager* doit repasser le document XML au *RunningManager* pour qu'il puisse vérifier s'il s'agit d'une *SuperSession* planifiée et puisse programmer ainsi son démarrage[3].
- *modifSession()* – Méthode qui reçoit en paramètre l'identificateur d'une *SuperSession* et permet de demander de modifier les informations de configuration de la *SuperSession*. Le *CreationManager* renvoie en réponse le document XML contenant les données de configuration de la *SuperSession*, enlevant temporairement le fichier de configuration correspondant du répertoire des *SuperSessions*.

Le deuxième groupe de méthodes permet de démarrer l'exécution des *SuperSessions*. Ces méthodes sont traitées par le sous-module *RunningManager* :

- *getSessions()* – Méthode permettant de demander au *Session Configuration Service* la liste des *SuperSessions* existantes ainsi que leur état (en exécution ou pas).
- *runSession()* – Méthode qui reçoit en paramètre l'identificateur d'une *SuperSession* et permet de demander au *Session Configuration Service* de lancer son exécution.

[1] Tout appel à des services Web est réalisé à travers le sous-module *SoapClient*.
[2] Dans l'Annexe B nous détaillons le format XML de ce fichier.
[3] Notons cependant que les *SuperSessions* planifiées ne sont pas encore supportées par notre prototype.

Pendant le traitement de cette méthode, le *RunningManager* demande au *SoapClient* de contacter les *Wrappers* des applications, utilisées en tant que support de cette *SuperSession*, en appelant la méthode *setSession()* définie dans l'interface de services Web de chaque application. Une fois que tous les *Wrappers* ont confirmé la mise en place de la *SuperSession*, il envoie ensuite une confirmation que la *SuperSession* a bien été démarrée. Si un des *Wrappers* ne réussit pas à exécuter les procédures nécessaires au démarrage la *SuperSession*, le processus est interrompu et le *RunningManager* renvoie en réponse un booléen négatif.

Le troisième et dernier groupe est également constitué de méthodes traitées par le *RunningManager*. Ces méthodes permettent de demander des informations concernant les *SuperSessions* en exécution :

- *getRunningSessions()* – Méthode qui renvoie en réponse la liste des *SuperSessions* en cours d'exécution.
- *getSesInfo()* – Méthode qui reçoit en paramètre l'identificateur d'une *SuperSession* et renvoie en réponse un document XML contenant les informations de configuration de cette *SuperSession*.

Comme indiqué précédemment, nous avons implémenté une application pour simuler le comportement de l'application Web qui donnera accès au *Session Configuration Service*. Cette application fait des appels à toutes les méthodes qui nous venons de décrire, à l'exception des deux dernières (*getRunningSessions()* et *getSesInfo()*) qui doivent en fait être invoquées par le *LClient*.

VI.3.4. *LClient*

LClient est une application Java qui se trouve dans le fichier "LClient.jar". Ce fichier contient notamment toutes les classes du paquetage "leica.client". La Figure VI.5 illustre le diagramme de classes de ce paquetage[1].

Lorsque *LClient* est exécuté, il crée initialement quatre de ses sous-composants : *EventNotInterface*, *ClientGUI*, *SessionManager* et *SoapClient*. Le *SessionManager* (par le biais du *SoapClient*) contacte le *Session Configuration Service* en appelant la méthode *getRunningSessions()* de son interface de services Web. La liste de *SuperSessions* en exécution, reçue en réponse, est repassée au *ClientGUI*. Ce dernier, étant le composant graphique de l'application, affiche cette liste dans laquelle l'utilisateur choisit une *SuperSession* qu'il désire rejoindre.

Une fois le choix réalisé, *LClient* récupère les informations de configuration de la *SuperSession* en appelant la méthode *getSesInfo()* du *Session Configuration Service*. Il demande ensuite à l'utilisateur de saisir des données d'identification (*i.e.* son identificateur, son nom, le rôle général et le mot de passe) pour pouvoir rejoindre la *SuperSession*. A gauche de la Figure VI.6, nous présentons l'interface graphique de départ du *LClient*.

Lorsque l'utilisateur appuie sur le bouton "Join SuperSession", *LClient* exécute les mêmes procédures que les *Wrappers* lors du démarrage d'une *SuperSession* : (i) la création du sous-module pour la gestion d'état (représenté par la classe *SS_class*) ; (ii) la création du sous-module pour l'exécution de la politique de collaboration de la *SuperSession* (représenté par la classe *PolicyManager*) ; et (iii) la mise en place du système de notification d'événements.

[1] Quelques classes utilitaires ont été enlevées du diagramme.

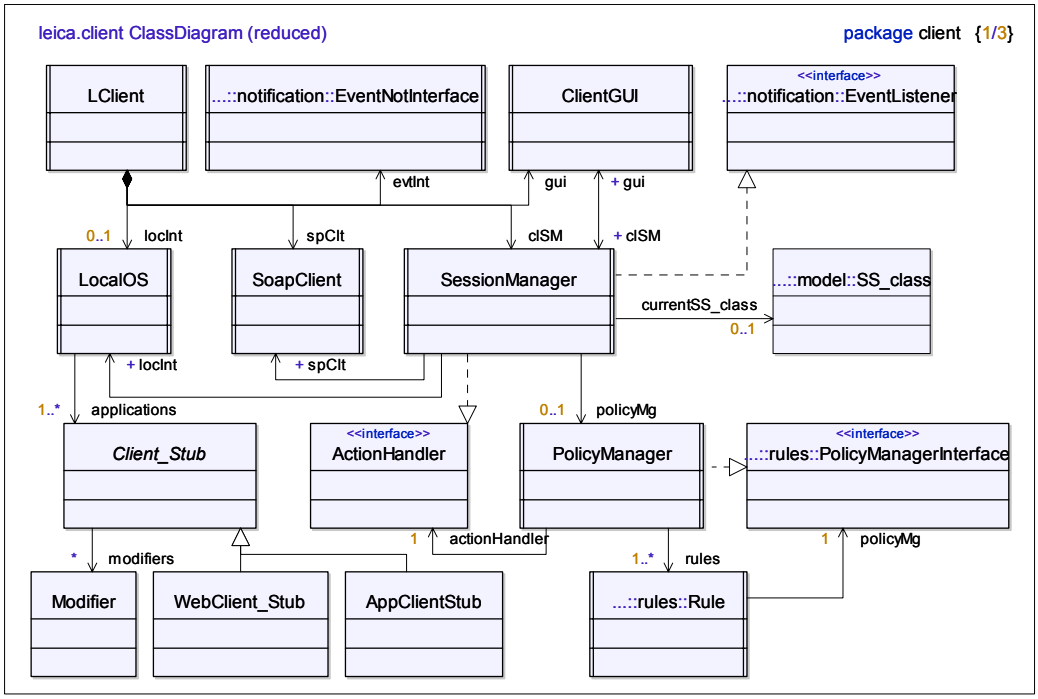

Figure VI.5 Diagramme de classes du paquetage leica.client

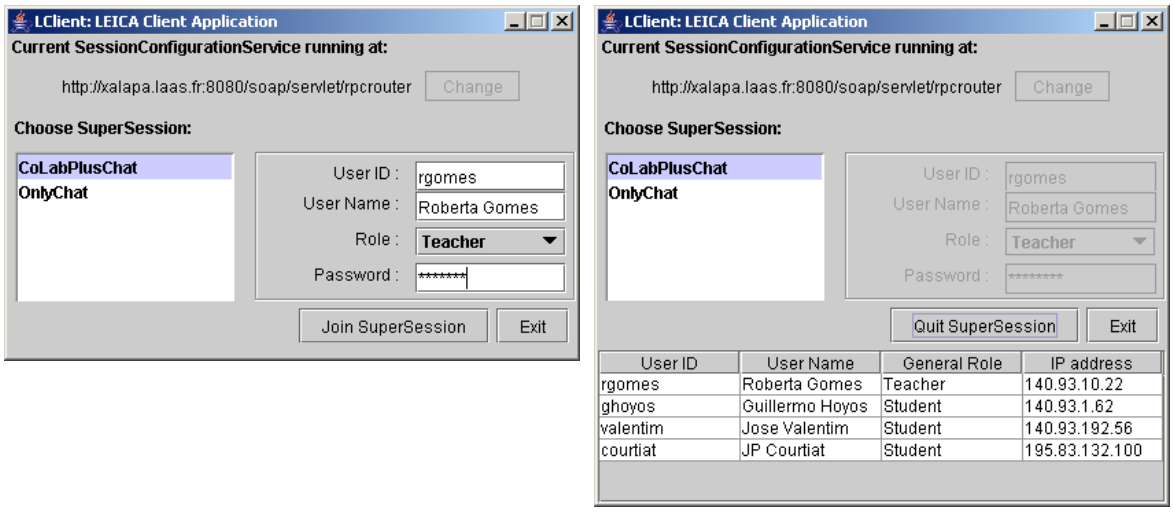

Figure VI.6 Interface graphique de l'application LClient

Tant que l'utilisateur fait partie d'une *SuperSession*, le *LClient* affiche l'état de la session (utilisateurs actuellement connectés). Le *SS_class* se charge ainsi de signaler au *ClientGUI* toute modification de l'état des utilisateurs connectés. A droite de la Figure VI.6, nous illustrons l'interface graphique du *LClient* pour un utilisateur connecté à une *SuperSession*.

Par rapport à l'exécution de la politique de collaboration, rappelons que le *LClient* peut être concerné par des règles spécifiant des requêtes d'exécution d'action du type "CONNECT" et "DISCONNECT" (dans les cas où les applications collaboratives ne sont pas capables de les traiter). Il demande ainsi à son *EventNotInterface* de s'abonner à des types d'événements qui peuvent sensibiliser ces règles. Il lui demande également de s'abonner aux événements de l'API de gestion pour que le *SS_class* puisse être informé des changements d'état de la *SuperSession*.

Pour traiter les requêtes d'exécution d'action du type "CONNECT" et "DISCONNECT", *LClient* crée des objets des classes *AppClient_Stub* et *WebClient_Stub* selon le type d'application utilisateur cliente (*stand-alone* ou *web-based*) respectif à chaque application. Nous trouvons dans le document XML de configuration de la *SuperSession* les informations concernant les paramètres utilisés lors de l'appel de ces applications. Normalement, les valeurs de ces paramètres peuvent être des constantes, ou extraites soit des paramètres de la requête d'exécution d'action, soit des composants d'état global de la *SuperSession*, mais ce dernier cas n'est pas encore supporté dans notre prototype.

VI.4. L'intégration d'outils de collaboration et déploiement de LEICA

Afin de tester les fonctionnalités implémentées dans notre prototype, nous avons intégré deux applications collaboratives. La première est *CoLab*, l'outil de navigation web coopérative que nous avons brièvement introduit dans la section IV.2.1.1. La deuxième application intégrée est *Babylon Chat* [Babylon], qui est un outil de messagerie instantanée multi-salons.

Nous avons choisi ces deux types d'application collaborative dans le but de mettre en œuvre le scénario d'intégration présenté dans la section IV.2.1.1 : un outil de navigation coopérative enrichi d'un outil de messagerie instantanée. Notre choix a été également guidé par le fait que *CoLab* est une application offrant une API d'accès tandis que *Babylon Chat* est un outil *open source*. Nous avons ainsi la possibilité de comparer le processus d'intégration d'applications selon qu'elles disposent ou pas d'une API.

VI.4.1. *CoLab*

CoLab est un outil développé au sein du groupe OLC du LAAS/CNRS et qui, dans sa version 2.0, définit une API qui rend possible son intégration à d'autres applications collaboratives. Le but de son concepteur a été de créer un outil simple qui représente une "brique" offrant des fonctionnalités spécifiques de collaboration. Par le biais de son API, elle peut être composée à d'autres outils, dans le but de développer un système plus complexe et adapté aux besoins des utilisateurs.

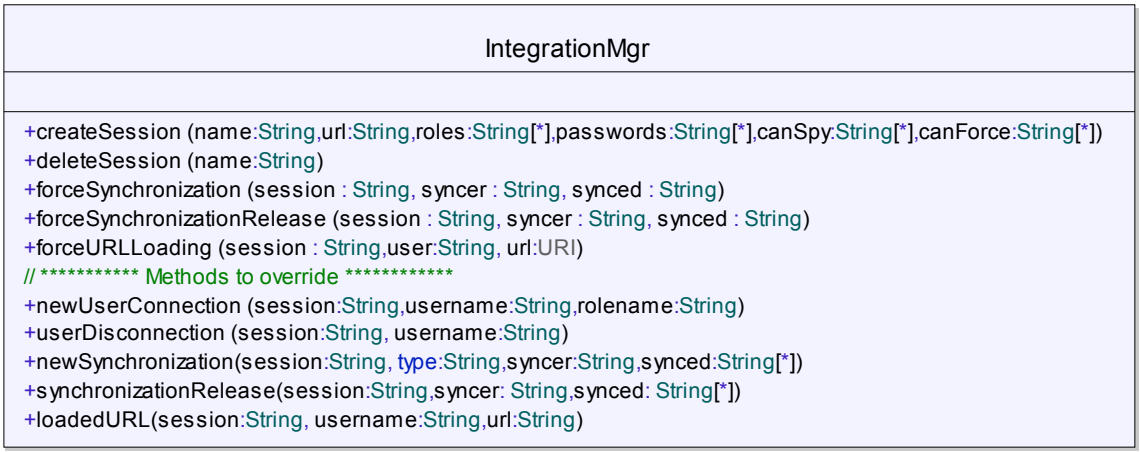

Figure VI.7 La classe IntegrationMgr de l'application CoLab

Le serveur *CoLab* consiste d'un *servlet* Java qui peut interagir avec son environnement par le biais d'un composant appelé *IntegrationMgr*. C'est donc ce dernier qui définit l'API d'accès au serveur *CoLab*. Dans la Figure VI.7 nous présentons la classe *IntegrationMgr* et les méthodes qui composent cette API.

Dans la première partie de cette API (les cinq premières méthodes listées dans la Figure VI.7) nous trouvons les méthodes permettant d'invoquer des procédures implémentées par le serveur. Dans la deuxième partie, nous avons des méthodes qui ne font aucun traitement spécial (leur listes d'instructions sont vides). Elles sont tout simplement invoquées par le serveur lorsque certaines opérations ont lieu pendant le déroulement de l'activité de navigation Web coopérative.

Pour pouvoir intégrer le serveur *CoLab* à une application, il faut créer une nouvelle classe appelée *IntegrationMgrImp* qui doit être une sous-classe de *IntegrationMgr*. Dans cette nouvelle classe, nous devons redéfinir les méthodes de la deuxième partie de l'API (en s'appuyant sur la technique d'*overriding*[1]) afin d'implémenter le traitement désiré lors de l'appel de ces méthodes.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://colab.laas.fr"
                      xmlns=" http://colab.laas.fr" elementFormDefault="qualified">
  <xsd:element name="spSspecs">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="initial_url" minOccurs="1" maxOccurs="1" type="xsd:uri">
          <xsd:annotation>
            <xsd:documentation>All connected users start browsing this initial URL.
              </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="role_privileges" minOccurs="1" maxOccurs="1" >
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="can_spy" minOccurs="0" maxOccurs="unbounded">
                <xsd:attribute name="from" type="xsd:string" use="required"/>
                <xsd:attribute name="to" type="xsd:string" use="required"/>
              </xsd:element>
              <xsd:element name="can_force" minOccurs="0" maxOccurs="unbounded">
                <xsd:attribute name="from" type="xsd:string" use="required"/>
                <xsd:attribute name="to" type="xsd:string" use="required"/>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure VI.8 Le fichier de données spécifiques de CoLab

Pour intégrer *CoLab* à LEICA, nous avons tout d'abord créé le fichier de données spécifiques ("CoLabsSspec.xsd"). Afin de définir ce fichier nous avons analysé la méthode *createSession()* de l'API de *CoLab* pour identifier les types de paramètres spécifiques nécessaires à la création d'une session de navigation Web coopérative. Nous avons ainsi repéré les attributs *url*, *canSpy* et *canForce* [2]. Le premier représente l'URL de départ pour tout nouvel utilisateur connecté à la session. Les deux derniers correspondent à des listes définissant des privilèges qu'un rôle peut avoir sur un autre rôle. Dans la Figure VI.8,

[1] Le phénomène où une nouvelle version d'une méthode (même signature) est redéfinie dans une sous-classe.
[2] Les autres paramètres (*name*, *roles*, et *password*) correspondent à des attributs communs à toutes les sessions spécifiques.

nous illustrons le fichier "CoLabsSspec.xsd" spécifiant ces paramètres spécifiques à *CoLab*.

Nous avons ensuite créé le fichier d'attributs et d'API ("CoLabAttrAPI.xml"), illustré dans la Figure VI.9. Dans ce fichier nous avons spécifié l'API événements/actions spécifiant les événements de type *Synchronization*, *DeSynchronization*, et *LoadURL* qui correspondent respectivement aux méthodes *newSynchronization()*, *synchronizationRelease()*, *loadedURL()* de l'API d'accès à *CoLab*. Nous avons également spécifié les actions de type *Synchronization*, *DeSynchronization*, et *LoadURL* qui correspondent respectivement aux méthodes *forceSynchronization()*, *forceSynchronizationRelease()*, et *forceURLLoading()* de l'API de *CoLab*.

```
<CA id="CoLab" wsRouter="http://banderilla.laas.fr:8090/soap/servlet/rpcrouter">
 <attributes>
  <name> CoLab </name>
  <description> Collaborative Web Browsing </description>
  <type> COOLABORATIVE_WEB_BROWSING </type>
  <maxRolePerUser> 1 </maxRolePerUser>
  <distribution> client-server </distribution>
  <userApplication>
   <webCallPattern url="http://colab.laas.fr/colab_login">
    <param name="loginname" value="action%Uid"/>   <param name="session" value="action%spSid"/>
    <param name="role" value="action%role"/>       <param name="password" value="action%pwd"/>
   </webCallPattern>
  </userApplication>
 </attributes>
 <API>
  <evtsAPI>
   <event type="Synchronization">
    <description> Notified when a new synchronization relationship is stablished between 2 users </description>
    <param name="synced">     <description> The user who looses browsing autonomy </description>
     <sequence base="string" size="*"/>
    </param>
    <param name="syncer" valueType="string"/>
   </event>
   <event type="DeSynchronization">
    <description> Notified when a new synchronization relationship is broken between 2 users </description>
    <param name="synced">     <description> The user who recovers browsing autonomy </description>
     <sequence base="string" size="*"/>
    </param>
    <param name="syncer" valueType="string"/>
   </event>
   <event type="LoadURL">
    <param name="user" valueType="string"/>   <param name="url" valueType="anyURI"/>
   </event>
  </evtsAPI>
  <actsAPI>
   <action type="Synchronization">
    <param name="synced" valueType="string">  <description> The user who looses browsing autonomy </description>
    </param>
    <param name="syncer" valueType="string"/>
   </action>
   <action type="DeSynchronization">
    <param name="synced" valueType="string"/>  <param name="syncer" valueType="string"/>
   </action>
   <action type="LoadURL">
    <param name="user" valueType="string"/>  <param name="url" valueType="anyURI"/>
   </action>
  </actsAPI>
  <gestionAPI>
   <action type="CONNECT" >
    <param name="Uid" valueType="string"/>  <param name="role" value="string"/>
    <param name="pwd" value="string"/>
   </action>
   <action type="DISCONNECT" >
    <param name="Uid" valueType="string"/>
   </action>
  </gestionAPI>
 </API>
</CA>
```

Figure VI.9 Le fichier d'attributs et d'API de CoLab

En ce qui concerne l'API de gestion, les événements de type CONNECT et DISCONNECT (prédéfinis pour toute application) correspondent aux méthodes *newUserConnection()* et *userDisconnection()*. Nous n'avons qu'à spécifier les actions de l'API de gestion (CONNECT et DISCONNECT) en indiquant que le serveur *CoLab* n'est pas capable de les traiter[1]. En fait, son API d'accès ne prévoit pas de méthode pour forcer la connexion/déconnexion d'un utilisateur.

Dans le fichier d'attributs et d'API nous avons encore défini les attributs de *CoLab*, notamment ceux décrivant son application cliente. Nous indiquons qu'il s'agit d'une application *web-based* ainsi que les paramètres utilisés lors de l'appel du navigateur web.

Dans la suite nous avons engendré le *Server Wrapper* adapté à *CoLab* et implémenté une sous-classe de *ApplicationInterface* (que nous avons appelée *ApplicationInterfaceImp*). Du côté de *CoLab*, nous avons implémenté la classe *IntegrationMgrImp* en suivant la méthode décrite précédemment. Finalement, nous avons lié les deux classes par les attributs *interfaceWithCoLab* et *interfaceWithLEICA*, et leurs méthodes ont été proprement accordées. Dans la Figure VI.10 nous détaillons les classes résultantes de ce processus d'intégration.

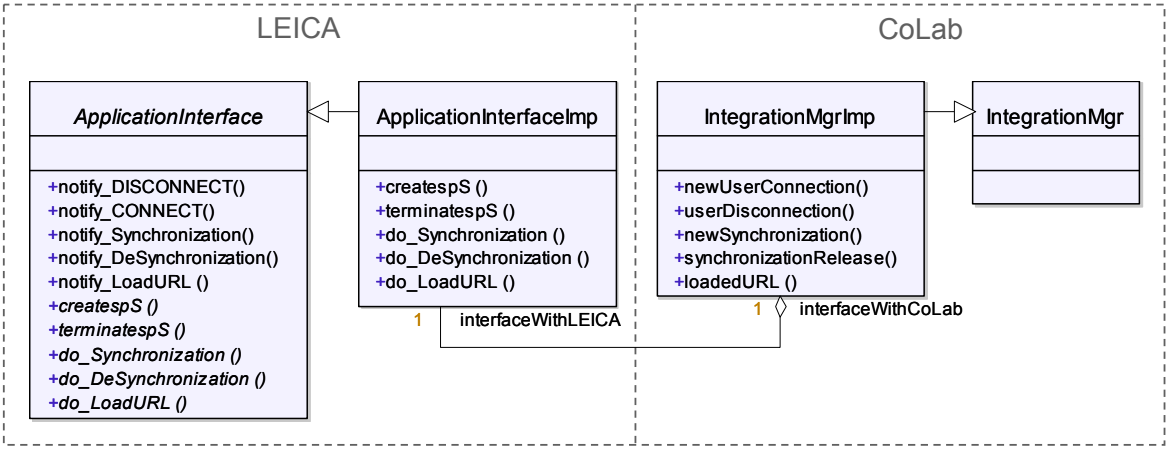

Figure VI.10 Les classes ApplicationInterface, ApplicationInterfaceImp et IntegrationMgrImp résultantes de l'intégration de CoLab à LEICA

Les efforts de développement employés lors de l'intégration de *CoLab* à LEICA ont été assez légers. Nous avons consacré quelques heures à décrire les fichiers et implémenter les classes, et en moins d'une journée, nous avions réalisé l'intégration. Bien évidemment, ce temps a été considérablement raccourci puisque nous n'avons pas eu à apprendre l'API d'accès à *CoLab*, que nous connaissions au préalable.

VI.4.2. *Babylon Chat*

Babylon Chat est une application client/serveur *open source* implémentée en Java. Elle représente un outil de messagerie instantanée permettant à des utilisateurs de communiquer librement entre eux ou de manière organisée : les utilisateurs peuvent créer et administrer des salons de chat afin de séparer les sujets de discussion. Outre le chat multi-salons, *Babylon Chat* implémente un tableau blanc partagé qui peut être utilisé par

[1] Cette indication est faite en plaçant la spécification de ces actions dans une balise spéciale "<gestionAPI>".

les clients connectés à une même session, mais nous n'avons pas exploité cette option dans le cadre de l'intégration avec LEICA.

Dans le but d'intégrer *Babylon Chat* à LEICA, nous avons modifié le serveur *Babylon* de façon à définir une API d'accès permettant d'interagir avec ce dernier. Nous avons commencé par la définition de l'API événements/actions de l'application : nous avons prévu un seul type d'événement, appelé *Entered_Room* (indiquant qu'un utilisateur vient d'entrer dans un salon de discussion), et deux types de requête d'exécution d'action, *User_Enter_Room* et *Group_Enter_Room* (permettant de déplacer un utilisateur, ou une liste d'utilisateurs, vers un salon de discussion ; si le salon indiqué n'existe pas, il est d'abord créé). Ensuite nous avons développé l'API d'accès au serveur composée par des méthodes correspondant à cette API événements/actions plus les événements de l'API de gestion. Quant aux actions de l'API de gestion, contrairement au cas de *CoLab*, nous avons pu modifier le serveur *Babylon* de façon qu'il puisse traiter l'action DISCONNECT de cette API.

Parallèlement à la définition de l'API d'accès, nous avons dû apporter une modification de fond concernant le serveur *Babylon*, qui à l'origine ne supporte qu'une seule session de Chat. Techniquement, chaque session doit correspondre à un serveur *Babylon*. Par conséquent, nous avons dû créer une nouvelle classe appelée *LEICABabylonServer* pour jouer le rôle de serveur de Chat multisessions. Cette classe doit lancer (et arrêter) des serveurs *Babylon* (des objets de la classe *babylonServer*) en vue de créer (et terminer) de sessions de Chat. Dans la Figure VI.11, nous illustrons les relations entre ces deux classes.

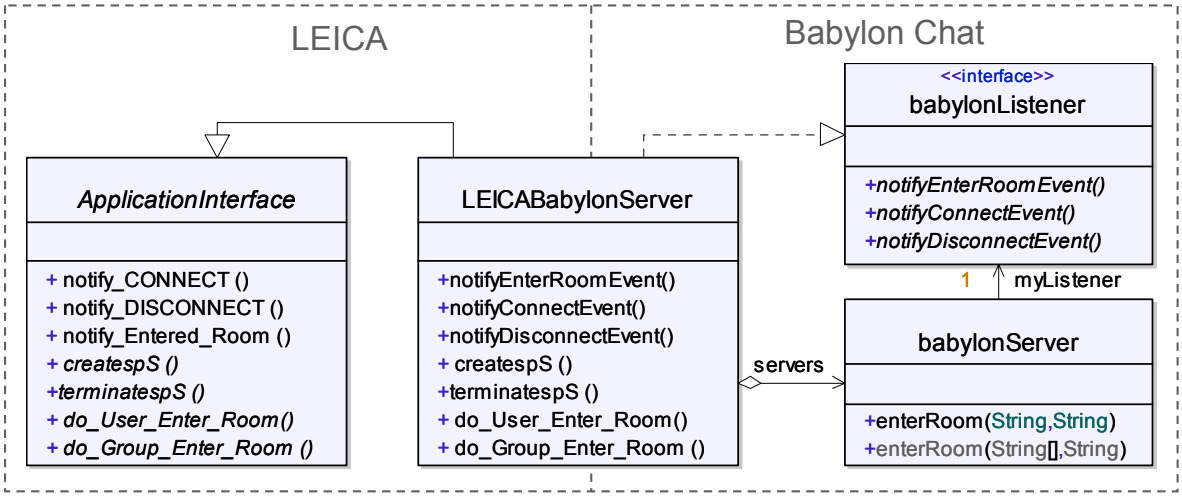

Figure VI.11 Les classes ApplicationInterface, LEICABabylonServer et l'interface babylonListener résultantes de l'intégration de Babylon Chat à LEICA

Comme nous l'avions fait pour *CoLab*, nous avons créé le fichier de données spécifiques et le fichier d'attributs et d'API de *Babylon Chat*. La Figure VI.12 illustre le premier fichier. Nous pouvons remarquer dans ce fichier que la seule information spécifique dont le Chat a besoin est un numéro de port.

Dans la Figure VI.13 nous illustrons le fichier d'attributs et d'API de *Babylon Chat*. Puisque le client de *Babylon Chat* est une application *stand-alone*, le mécanisme d'appel local de cette application peut varier selon le système d'exploitation déployé sur le poste

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://www.laas.fr/~rgomes/Babylonchat"
                      xmlns="http://www.laas.fr/~rgomes/Babylonchat"
                      elementFormDefault="qualified">
  <xsd:element name="spSspecs">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Port" minOccurs="1" maxOccurs="1" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>One server per TCP port.</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure VI.12 Le fichier de données spécifiques de Babylon Chat

```
<CA id="Chat" wsRouter="http://xalapa.laas.fr:8080/soap/servlet/rpcrouter">
 <attributes>
  <name> Chat </name>
  <description> MultiRoom Chat </description>
  <type> CHAT </type>
  <maxRolePerUser> 0 </maxRolePerUser>
  <distribution> client-server </distribution>
  <userApplication>
   <appCallPattern system="windows" call="java babylon">
    <modifier name="-username" value="action%username" /> <modifier name="-servername" value="action%server" />
    <modifier name="-portnumber" value="action%port" /> <modifier name="-autoconnect" />
    <modifier name="-chatroom" value="action%chatroom" /> <modifier name="-nopasswords" />
    <modifier name="-locksettings" /> <modifier name="-hidecanvas" />
   </appCallPattern>
   <appCallPattern system="unix" call="java babylon">
    <modifier name="-username" value="action%username" /> <modifier name="-servername" value="action%server" />
    <modifier name="-portnumber" value="action%port" /> <modifier name="-autoconnect" />
    <modifier name="-chatroom" value="action%chatroom" /> <modifier name="-nopasswords" />
    <modifier name="-locksettings" /> <modifier name="-hidecanvas" />
   </appCallPattern>
  </userApplication>
 </attributes>
 <API>
  <evtsAPI>
   <event type="Entered_Room">
    <param name="userName" valueType="string"/> <param name="roomName" valueType="string"/>
   </event>
  </evtsAPI>
  <actsAPI>
   <action type="User_Enter_Room">
    <param name="userName" valueType="string"/> <param name="roomName" valueType="string"/>
   </action>
   <action type="Group_Enter_Room">
    <param name="usersName"> <sequence base="string" size="*"/> </param>
    <param name="roomName" valueType="string"/>
   </action>
   <action type="DISCONNECT">
    <param name="Uid" valueType="string"/>
   </action>
  </actsAPI>
  <gestionAPI>
   <action type="CONNECT">
    <param name="Uid" valueType="string"/> <param name="server" valueType="string"/>
    <param name="port" valueType="string"/> <param name="chatroom" valueType="string"/>
   </action>
  </gestionAPI>
 </API>
</CA>
```

Figure VI.13 Le fichier d'attributs et d'API de Babylon Chat

utilisateur. Ainsi, contrairement à *CoLab*, nous avons spécifié deux formats d'appels de l'application cliente (le premier pour les systèmes *Windows*, et le deuxième pour *Unix*[1]). Nous pouvons également observer dans ce fichier l'événement *Entered_Room* et les actions *User_Enter_Room* et *Group_Enter_Room* définis dans l'API événements/actions du Chat. En ce qui concerne l'API de gestion, la différence par rapport à CoLab se trouve dans l'action DISCONNECT, traitée par le serveur lui-même[2].

En utilisant ces deux fichiers, nous avons par la suite engendré le *Server Wrapper* adapté au Chat. Si nous revenons sur la Figure VI.11, nous pouvons remarquer les détails sur la *ApplicationInterface*, créée lors de la génération du nouveau *Wrapper*.

Pour intégrer le *Wrapper* à *Babylon Chat*, nous n'avons pas créé une nouvelle classe héritant de *ApplicationInterface*, comme nous l'avions fait pour *CoLab*. En fait, nous avons défini *LEICABabylonServer* directement en tant que sous-classe. Notons que, puisque Java ne supporte pas d'héritage multiple, nous avions été obligé de concevoir deux classes distinctes dans le cas de *CoLab*.

Finalement nous avons fini d'implémenter la classe *LEICABabylonServer* en faisant les liaisons entre les méthodes de l'API de *Babylon Chat* et celles de la *ApplicationInterface*. Ce dernier effort de développement a été celui ayant consommé le moins de temps pendant le processus d'intégration de *Babylon Chat* à LEICA, qui au total a duré quelques journées. La plupart du temps a été consacré à l'étude du code source du serveur *Babylon* et à l'implémentation des méthodes permettant d'interagir avec lui.

VI.4.3. La *SuperSession* "CoLabPlusChat"

Une fois les deux applications, *CoLab* et *Babylon Chat* (par simplicité nous lui avons associé l'identificateur *Chat*), ont été intégrées, l'étape suivante a consisté à créer une *SuperSession* implémentant le comportement décrit dans le scénario d'intégration de la section IV.2.1.1. Nous avons créé manuellement le fichier de configuration de la *SuperSession* que nous avons intitulé "CoLabPlusChat".

Dans le *GSMinfo* nous avons simplement fourni les informations de contexte ainsi qu'une liste de rôles généraux (juste à titre illustratif) composée de deux rôles : "Teacher" et "Student". Dans le *IAinfo* nous avons spécifié, pour chacune des applications, une session spécifique. Dans la Figure VI.14, nous avons la première partie du fichier de configuration de notre *SuperSession* montrant ces deux éléments.

Dans la Figure VI.15, nous avons la deuxième partie du fichier de configuration où nous spécifions la politique de collaboration de la *SuperSession*. La première règle politique, vue dans la figure, définit la connexion initiale des utilisateurs aux applications collaboratives. Selon ce que nous avons décrit dans le scénario d'intégration, les utilisateurs connectés à *CoLab* et faisant partie d'un même arbre de navigation doivent se trouver dans le même salon de discussion du *Chat*. Mais rappelons qu'initialement un utilisateur qui se connecte à *CoLab* se trouve dans l'état asynchrone – il représente à lui seul un arbre de navigation dont il est le seul nœud (la racine). Ainsi lorsqu'il est connecté au *Chat* il rentre directement dans un salon de discussion qui porte le même nom que son identificateur (c'est donc l'utilisateur à la racine d'un arbre qui donne le nom du salon de discussion associé).

[1] Comme il s'agit d'une application Java, les deux formats d'appels sont en fait identiques.

[2] Cette indication est faite en plaçant la spécification de l'action dans la balise "<actsAPI>", avec toutes les autres actions de l'API événement/actions.

```
<SuperSession id="CoLabPlusChat" >
  <GSMInfo>
    <context>
      <name> CoLab plus Chat </name>
      <description> Cooperative web navigation integrated to a Chat application. </description>
    </context>
    <scheduling>  <starting> null </starting>  <duration> null </duration> </scheduling>
    <roles>
      <role id="Teacher">
        <description> The teacher of a SuperSession. </description>
        <membership> <password> teacherp </password> </membership>
        <maxSimUsers> 1 </maxSimUsers>
        <rights assignment="all"> chair </rights>
      </role>
      <role id="Student">
        <description> Students of a SuperSession. </description>
        <membership>  <password> studentp </password> </membership>
        <maxSimUsers> -1 </maxSimUsers>
        <rights> null </rights>
      </role>
    </roles>
  </GSMInfo>
<IAInfo>
    <CA id="CoLab">
      <attributes>  ...  </attributes>
      <spS id="CoLabSession">
        <scheduling>
          <starting> null </starting> <duration> null </duration>
        </scheduling>
        <roles type="automatic">
          <role id="Navigable">
            <membership>   <password> ppp </password>   </membership>
            <baseRole> Teacher </baseRole>   <baseRole> Student </baseRole>
          </role>
        </roles>
        <spSspecs>
          <initial_url> http://www.laas.fr </initial_url>
          <role_privileges>  </role_privileges>
        </spSspecs>
      </spS>
    </CA>
    <CA id="Chat">
      <attributes> ...  </attributes>
      <spS id="ChatSession">
        <scheduling>
          <starting> null </starting>  <duration> null </duration>
        </scheduling>
        <roles type="none"/>
        <spSspecs>
          <Port> 55555 </Port>
        </spSspecs>
      </spS>
    </CA>
  </IAInfo>
  ...
```

Figure VI.14 Le GSMinfo et le IAinfo de la SuperSession "CoLabPlusChat"

Les deux dernières règles illustrées dans la Figure VI.15 permettent d'assurer que les membres d'un même arbre de navigation Web appartiennent toujours au même salon de discussion. Vu que *CoLab* est capable de notifier un événement à chaque fois qu'une relation de synchronisation est créée entre deux utilisateurs (événement de type *Synchronization*), il faut simplement associer à cette notification une requête d'exécution d'action destinée au *Chat* pour qu'il déplace l'utilisateur (et les fils de l'arbre dont il était la racine) qui vient de devenir synchrone (*i.e.* qui a perdu son autonomie de navigation) dans le salon de discussion associé à l'arbre auquel il a été raccordé. La deuxième règle de la figure spécifie ce comportement, tandis que la troisième spécifie le comportement symétrique correspondant à la désynchronisation d'un utilisateur.

```
…
 <rules>
   <rule>
    <event from="LEICA" type="CONNECT" id="1">
    </event>
    <actions>
     <action to="CoLab" type="CONNECT">
       <param name="spSid" > <value> CoLabSession   </value> </param>
       <param name="Uid">   <value> %1.Uid         </value> </param>
       <param name="role">  <value> Navigable      </value> </param>
       <param name="pwd">   <value> ppp            </value> </param>
     </action>
     <action to="Chat" type="CONNECT">
       <param name="spSid" > <value> ChatSession    </value> </param>
       <param name="Uid">   <value> %1.Uid         </value> </param>
       <param name="server"> <value> xalapa.laas.fr </value> </param>
       <param name="port">  <value> 55555          </value> </param>
       <param name="chatroom"> <value> %1.Uid      </value> </param>
     </action>
    </actions>
   </rule>
   <rule>
    <event from="CoLab" type="Synchronization" id="1" >  </event>
    <actions>
     <action to="Chat" type="Group_Enter_Room">
       <param name="usersName"> <values> %1.synced </values></param>
       <param name="roomName">  <value> %1.syncer  </value></param>
       <param name="spSid">     <value> ChatSession </value></param>
     </action>
    </actions>
   </rule>
   <rule>
    <event from="CoLab" type="DeSynchronization" id="1" >  </event>
    <actions>
     <action to="Chat" type="Group_Enter_Room">
       <param name="usersName"> <values> %1.synced  </values></param>
       <param name="roomName"> <value> %1.synced[0] </value> </param>
       <param name="spSid" > <value> ChatSession    </value> </param>
     </action>
    </actions>
   </rule>
 </rules>
</SuperSession>
```

Figure VI.15 La politique de collaboration de la SuperSession "CoLabPlusChat"

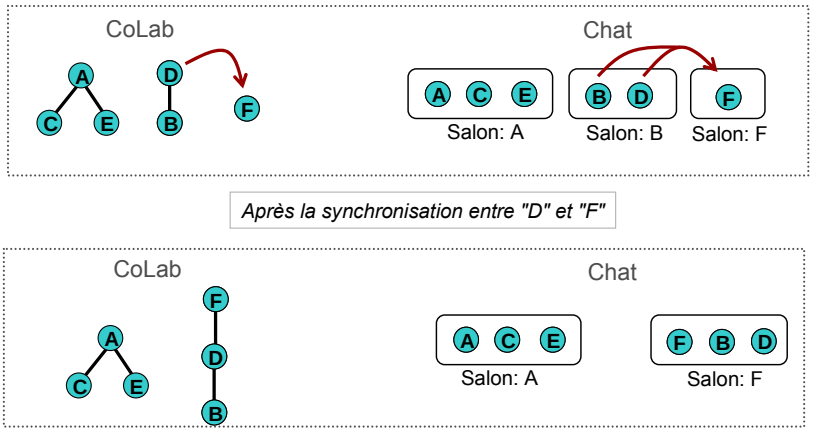

Figure VI.16 La synchronisation d'un utilisateur dans CoLab et son déplacement vers un salon de discussion dans le Chat

Afin d'illustrer ce comportement dynamique, considérons le scénario illustré en haut de la Figure VI.16 où nous avons 6 utilisateurs connectés à la *SuperSession*. Trois arbres de navigation se sont formés dans la session spécifique de *CoLab* et trois salons de discussions sont définis dans la session spécifique du *Chat*. Supposons que l'utilisateur "D" se synchronise avec l'utilisateur "F". Lorsque cet événement est notifié par *CoLab*, comme résultat de l'exécution de la politique de collaboration, le *Chat* exécute une action

en déplaçant l'utilisateur "D", ainsi que tous ses fils ("B" dans ce cas), dans le salon de discussion nommé "F".

VI.4.4. L'environnement d'exécution

Les différents modules de LEICA, comprenant les deux applications intégrées, ont été déployés sur des ordinateurs raccordés au réseau local du LAAS-CNRS, tel que nous l'illustrons dans la Figure VI.17. Nous avons exécuté la *SuperSession* "ColabPlusChat" et connecté 4 utilisateurs différents. Les utilisateurs ont réalisé plusieurs synchronisations et désynchronisations dans l'application cliente de *CoLab* afin d'observer leur déplacement automatique entre les salons de discussion du *Chat*.

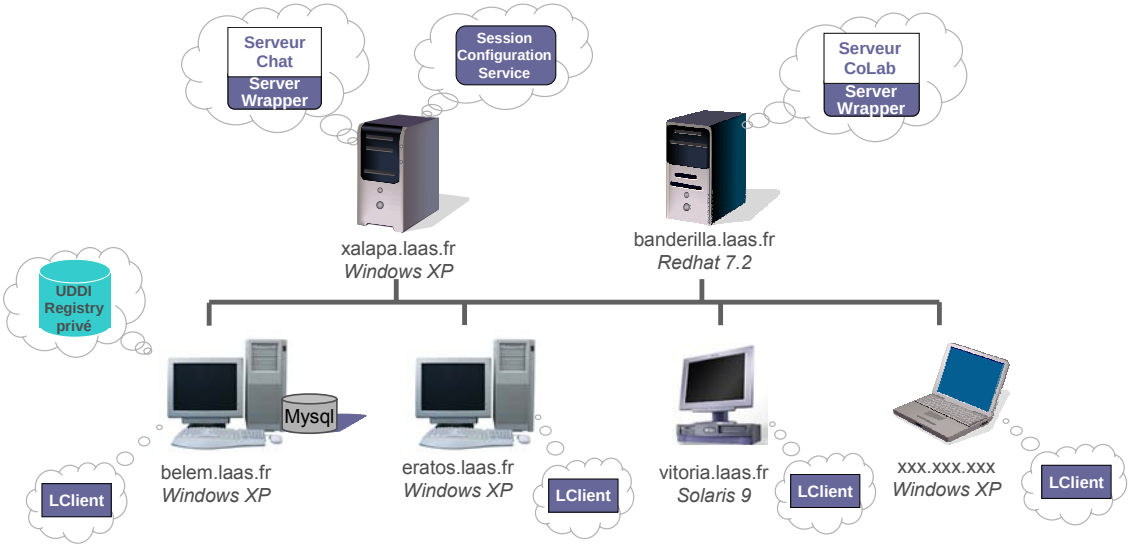

Figure VI.17 L'environnement d'exécution de LEICA

VI.4.5. Quelques considérations sur le prototype

Malgré le fait que ce premier prototype ne présente qu'une petite partie de l'ensemble des fonctionnalités définies par LEICA, il nous a permis effectivement de mettre en œuvre les principaux concepts de l'approche d'intégration proposée :

a) L'intégration faiblement couplée

Cette approche a beaucoup contribué à la simplicité et à la faisabilité du processus d'intégration proposé. Une intégration indépendante de chaque application, sans que l'une ne soit au courant des autres, rend l'environnent beaucoup plus flexible. De plus, l'autonomie de chaque application est préservée.

Par contre, un des inconvénients de cette approche est relatif au fait que l'environnement hôte utilisateur doit être préalablement configuré en terme d'installation. Les applications clientes et les applications P2P qui seront utilisées en tant que support d'une *SuperSession* doivent être mises en place avant que l'utilisateur n'exécute le *LClient*. Cette contrainte n'est pas vraiment critique puisque dans le cas où les utilisateurs iraient utiliser ces applications de manière indépendante (c'est-à-dire sans être intégrées à LEICA) ils se trouveraient exactement dans la même situation. Néanmoins, nous sommes convaincus qu'un déploiement dynamique complet des applications clientes (sans installation préalable) serait beaucoup plus intéressant.

b) L'utilisation des technologies de services Web

Les services Web ont joué un rôle important dû à leur aspect faiblement couplé et à leur simplicité de déploiement. La technologie s'est avérée très adaptée à l'implémentation des interactions basée sur messages entre les modules de LEICA. En contre partie, nous avons remarqué, en termes de temps de réponse, le coût de l'emploi de cette technologie. Nous avons réalisé quelques mesures pour calculer de façon approximative le temps rajouté par la couche de services Web lors d'un appel à une méthode. Par exemple, dans le cas des méthodes de l'interface de services Web du *Session Configuration Service*, nous avons eu comme moyenne une valeur de 35 ms[1], alors que, en simulant les appels aux méthodes par *socket* TCP directement, le temps rajouté par la couche de transport a été en moyenne inférieur à 10ms[2].

c) Le système de notification d'événements

L'utilisation de *Scribe* pour implémenter le système de notification d'événements a été très convenable. Il présent une API très simple à utiliser et extensible (avec par exemple un mécanisme simple pour implémenter l'ordonnancement des messages envoyés).

Au niveau des performances, nous avons un coût initial important imposé par *Pastry* lors de la connexion d'un nouveau nœud à un réseau logique. Cette connexion prend entre 5 et 10 secondes pour s'effectuer. Néanmoins, l'envoi de notifications d'événement s'est montré très performant. Juste à titre illustratif, nous avons conduit une petite expérimentation dans un LAN de 5 nœuds (simulant 5 *Wrappers*) connectés au même réseau logique. Chaque nœud s'est mis à publier des événements tandis que tous les autres, abonnés au sujet correspondant, les recevaient. La latence moyenne entre l'instant de publication d'un événement et sa réception a été de l'ordre de 300ms. Nous n'avons pas vérifié le comportement de cette latence lors du passage à l'échelle, ni en fonction du nombre de nœuds, ni en fonction du nombre de groupes multicast. Mais nous avons consulté plusieurs résultats dans [Castro-02] où nous pouvons constater une performance raisonnable en terme de délai et de charge du réseau par rapport à IP multicast (*Scribe* étant entre 1.59 et 2.2 plus lent que ce dernier).

d) L'exécution de la politique de collaboration

La répartition de l'exécution des règles politiques entre les différents *Wrappers* (et les *LClient*s) connectés à une *SuperSession* peut bien évidement contribuer au passage à l'échelle par rapport au nombre de règles définies dans sa politique de collaboration. Néanmoins, nous devons quand même considérer le fait qu'un même *Wrapper* ou un *LClient* peut être obligé d'exécuter plusieurs règles en même temps (définies dans la politique de collaboration d'une même *SuperSession*, ou de différentes *SuperSessions*). Vu que dans notre implémentation chaque règle représente un *thread* Java, un nombre important de règles en exécution pourrait compromettre la performance du sous-module d'exécution de la politique de collaboration (*i.e.* le *PolicyManager*).

Ainsi nous avons réalisé une expérimentation afin de vérifier le temps moyen pris par le *PolicyManager* pour exécuter l'ensemble des règles lorsqu'un événement attendu (un événement qui peut sensibiliser une règle) est notifié. Tout d'abord nous avons défini une règle simple associant une notification d'événement à une requête d'exécution d'action.

[1] Pour réaliser les mesures nous avons exécuté les appels localement et enlevé le traitement correspondant aux appels.

[2] Remarquons que la précision de chaque mesure s'est trouvée (approximativement) limitée à 15 ms due à l'implémentation du système d'exploitation *Windows XP* sur lequel nous avons réalisé ces mesures.

Ensuite nous avons programmé le *PolicyManager* pour exécuter un nombre paramétrable de copies de cette règle. L'expérimentation consistait à exécuter le *PolicyManager* différentes fois en variant le nombre de règles (de 10, 100, 1000, 2000, 3000, 4000 et 5000) et de simuler (10 fois pour chaque exécution) une notification d'événement en mesurant le temps nécessaire pour que toutes les règles en exécution soient sensibilisées[1]. Dans la Figure VI.18 nous avons les moyennes des résultats obtenus pour 5 répétitions de l'expérimentation.

Si on considère que (i) les *SuperSessions* présenteront très difficilement des politiques de collaboration composées par des centaines de règles, et si on estime que (ii) des règles plus complexes (en utilisant de *Earliests* et des *Latests*) soient une vingtaine de fois plus lentes à exécuter (ce que nous croyons être une estimation vraiment pessimiste), le délai d'exécution reste toujours raisonnable (de l'ordre de 100 ms).

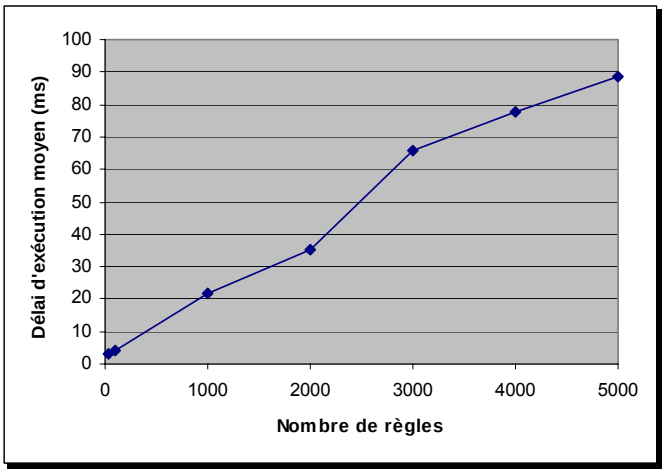

Figure VI.18 Le délai moyen pris par le PolicyManager pour exécuter et sensibiliser toutes les règles lors d'une notification d'événement, en fonction du nombre de règles

VI.5. Conclusions

Dans ce chapitre, nous avons présenté en détail l'implémentation du prototype actuel de LEICA. Nous avons tout d'abord présenté et justifié les choix technologiques, notamment en ce qui concerne le déploiement des interfaces de services Web et du système de notification d'événement basé sur le paradigme *publish*/*subscribe*.

Nous avons décrit les modules de LEICA implémentés pour le prototype, qui ne supporte actuellement que l'intégration d'applications client/serveur. En détaillant l'*API Factory*, nous avons montré dans la pratique comment s'effectuent la génération dynamique du *Server Wrapper* et son intégration au serveur de l'application. Concernant ce dernier, nous avons donné des détails (i) sur son interface de services Web, (ii) sur les classes implémentées pour l'exécution de la politique de collaboration (qui pour le moment ne supporte pas les *Latests*, *Earliests* et *Triggers*), et (iii) sur la mise en place du système de notification d'événements. Une description du *Session Configuration Service*, en fonction des méthodes rendues disponibles dans son interface de services Web, et du *LClient* ont été également présentées.

[1] L'ordinateur utilisé est un PC Pentium 2.9 GHz, 1Go de Ram, sous Windows XP.

Nous avons ensuite détaillé le processus d'intégration de deux applications collaboratives à LEICA : *CoLab* et *Babylon Chat*. Pour chacune de ces applications, nous avons décrit le processus de génération dynamique des *Server Wrappers* respectifs ainsi que les efforts de développement réalisés lors de leurs intégrations aux serveurs.

Finalement, nous avons décrit la *SuperSession*, appelée "CoLabPlusChat", créée dans le but de tester le prototype. Malgré les limitations du prototype, l'exécution de cette *SuperSession*, ainsi que quelques tests et mesures réalisées, nous ont permis de faire quelques remarques intéressantes sur les principaux concepts de l'approche d'intégration que nous avons développée dans LEICA.

Chapitre VII

Conclusion générale et perspectives

VII.1. Bilan des travaux réalisés

Les travaux de recherche développés dans cette thèse s'inscrivent dans le domaine du "Travail Coopératif Assisté par Ordinateur", ou TCAO. Plus précisément, nous avons proposé une approche générale pour créer des environnements collaboratifs s'appuyant sur une intégration d'applications collaboratives (ou collecticiels) existantes. En suivant cette approche, nous avons développé LEICA, un environnement faiblement couplé qui, par le biais de *Wrappers* (sortes de médiateurs), rend effectivement possible l'intégration de telles applications. Dans le contexte de ce que nous avons défini comme étant une *SuperSession*, nous disposons alors des outils pour mettre en place une activité collaborative globale constituée de différentes applications intégrées à LEICA utilisées en parallèle, de manière complémentaire et coordonnée.

Une des originalités de nos travaux réside dans notre proposition d'intégrer des applications développées par des tiers. Un aspect important qui a permis l'intégration de ces applications est l'approche faiblement couplée adoptée par LEICA. Une telle approche nous permet de considérer les applications sous la forme de boites noires, sans avoir besoin de connaître les détails de leur fonctionnement interne. Aucun changement des structures internes des applications ne s'est également avéré nécessaire pour réaliser l'intégration souhaitée.

Dans la pratique, ceci nous a conduit à définir comme cible d'applications à intégrer, des applications collaboratives disposant d'APIs, ces APIs étant essentielles dans le processus de couplage des *Wrappers* aux applications. En effet, le niveau d'intégration obtenu par LEICA dépend directement de l'API disponible. Le développement de collecticiels qui puissent répondre au mieux aux besoins des utilisateurs implique des niveaux de complexité très importants, qui sont notamment dus à méconnaissance à priori des besoins des utilisateurs. En conséquence, les développeurs ont et auront dans l'avenir tout intérêt à concevoir des applications qui représentent des briques de base offrant des fonctionnalités collaboratives bien spécifiques, qui puissent ensuite être intégrées à d'autres applications par l'intermédiaire d'une API qui soit la plus flexible possible.

Même si le développement de LEICA a été motivé par l'idée de disposer d'un environnement intégré permettant de combiner différentes fonctionnalités collaboratives, nous pourrions également envisager LEICA en tant que solution pour réaliser l'interopérabilité entre collecticiels équivalents. L'intérêt serait alors de permettre à des utilisateurs qui doivent travailler ensemble d'avoir une certaine liberté de choix vis-à-vis des collecticiels utilisés en support de leur activité collaborative. Deux utilisateurs pourraient, par exemple, éditer ensemble un même document en utilisant chacun l'éditeur partagé de sa préférence. Néanmoins, ce scénario d'intégration pourrait exiger un niveau

beaucoup plus fin dans les types de messages qu'une application devrait être capable d'échanger. Dans le cas de l'édition d'un document, il faudrait que chaque modification réalisée sur le document dans une application soit notifiée à l'autre application. Si nous considérons que plus une API est détaillée, plus il est difficile de la développer, il ne faut vraisemblablement pas avoir trop d'attentes concernant le niveau des APIs rendues disponibles par les applications.

Malgré le côté faiblement couplé de son infrastructure d'intégration, LEICA offre beaucoup plus qu'une simple intégration de surface. La spécification d'une politique de collaboration, constituée d'un ensemble de règles, pour chaque *SuperSession* permet de définir la manière dont les applications (utilisées en tant que support d'une *SuperSession*) doivent réagir aux notifications d'événement. L'approche graphique choisie pour la spécification des règles politiques permet au créateur de la *SuperSession* de donner une sémantique plus intuitive à l'intégration. Par ailleurs, à l'aide des réseaux de Pétri et leurs extensions, nous avons pu, de notre côté, définir la sémantique précise de ces règles politiques.

Pour mener à bien la conception de LEICA, nous sommes tout d'abord passés par une phase d'analyse et de projet de son architecture. Nous avons ainsi défini formellement l'architecture de l'environnement d'intégration en utilisant le profil UML/SDL supporté par l'outil *TAU G2* de *Telelogic*. Nous avons défini une méthode de spécification de l'architecture qui s'appuie sur un ensemble de diagrammes et avons validé cette architecture par simulation en comparant les traces de simulation représentées sous la forme de diagrammes de séquence avec les scénarios utilisateurs conçus préalablement. Outre la validation de l'architecture, cette modélisation UML a été très importante pour documenter LEICA, tout en montrant la complexité d'un tel environnement d'intégration réparti. Cette documentation a également servi de pont entre l'approche générale d'intégration proposée au départ et l'implémentation présentée dans le dernier chapitre.

L'implémentation du prototype de LEICA a été la dernière contribution de nos travaux. Malgré que nous n'ayons pas été en mesure, par manque de temps, d'implémenter tous les modules prévus initialement, ce prototype nous a permis de vérifier la viabilité de l'approche d'intégration proposée. Deux applications collaboratives ont été intégrées et plusieurs tests ont été réalisés afin de pouvoir tirer quelques premières conclusions sur la performance de l'environnement.

VII.2. Les perspectives de nos travaux

Notre premier objectif consiste à poursuivre nos travaux pour une mise en œuvre complète de LEICA (par rapport à la spécification de son architecture), comprenant l'implémentation

(i) des modules nécessaires à l'intégration des applications P2P et multiserveurs ;

(ii) de l'application Web d'accès au *Session Configuration Service* ;

(iii) du support aux règles politiques composées par des *Earliests* et *Latests*.

L'intégration d'autres applications collaboratives à LEICA, notamment des applications commerciales (par exemple *Skype*), est bien évidement envisageable et envisagée. Il nous faudra également mettre en place les mécanismes d'ordonnancement et d'estampillage temporel (s'appuyant sur la synchronisation des horloges physiques), afin d'éviter les incohérences potentielles dues à la répartition (décrites dans le paragraphe V.2.4).

Au delà de la poursuite de l'implémentation de LEICA, plusieurs autres perspectives s'offrent à nous pour l'approfondissement de notre travail :

– *L'évolutivité des applications collaboratives*

Remarquons que l'approche d'intégration adoptée par LEICA est bien dépendante des APIs des applications collaboratives :

(i) un *Wrapper* est généré dynamiquement pour une application en s'appuyant sur des descriptions XML de son API ;

(ii) la spécification de la politique de collaboration d'une *SuperSession* dépend directement de l'API événements/actions des applications.

Par conséquent, une fois que l'on a intégré une application à LEICA et que les *SuperSessions* qui les utilisent ont été créées, on pourrait être amené à modifier leurs API événements/actions. Pourtant, LEICA impose des contraintes concernant l'évolution des applications collaboratives vis-à-vis de leurs APIs. En effet, l'évolution d'une application doit être reportée dans le contexte du *Wrapper*. Le surcoût de cette tâche ne constitue pas vraiment un problème puisque engendrer un nouveau *Wrapper* (et le raccorder à l'application) pour prendre en compte des modifications de l'API d'une application est relativement simple.

Néanmoins, dans la définition des politiques de collaboration des *SuperSessions*, il est possible qu'il existe des dépendances vis-à-vis des APIs des applications. Ainsi, si au lieu d'uniquement ajouter de nouveaux types d'événements et d'actions dans l'API, ces modifications concernent également des modifications dans les types originaux d'événements et d'actions, cela pourrait entraîner des problèmes lors de l'exécution de la politique de collaboration d'une *SuperSession*. Par exemple, quand on modifie l'API d'une application, on peut supprimer un type d'action spécifique qui pourrait être utilisé dans une règle politique d'une *SuperSession*. La sensibilisation de cette règle produirait alors une requête d'exécution d'action qui ne serait plus supportée par l'application.

Une solution envisageable pour résoudre ce problème serait de mettre en place un mécanisme qui permette de suivre les modifications apportées à l'API d'une application et d'identifier automatiquement quelles sont les *SuperSessions* affectées par ces modifications. Cela pourrait guider les créateurs des *SuperSessions* concernées dans la modification de ces *SuperSessions*, de façon que leurs politiques de collaboration puissent prendre en compte les modifications de ces APIs.

– *Vers une infrastructure complètement ouverte*

Un aspect intéressant de LEICA concerne son architecture hybride pour mettre en œuvre une intégration faiblement couplée. L'utilisation des technologies de services Web pendant les étapes initiales (intégration d'une application, création et lancement des *SuperSessions*) nous a permis d'éviter les problèmes de performance associés au protocole SOAP. Néanmoins, l'utilisation d'un service de notification d'événements (s'appuyant sur le paradigme *publish*/*subscribe*) pendant l'exécution d'une *SuperSession* impose l'emploi dans les *Wrappers* d'une technologie qui n'est pas normalisée. Par conséquent, nous contraignons le processus d'intégration, tandis qu'une infrastructure s'appuyant exclusivement sur des standards rendrait LEICA plus ouvert ; les développeurs pourraient alors développer eux-mêmes leurs *Wrappers* (qui pourrait même être implémentés en utilisant d'autres langages que Java).

Un effort important a été déployé pour proposer des solutions permettant de compacter le format XML et, plus spécifiquement, pour optimiser la transmission et le codage de messages SOAP. Nous avons, par exemple, l'ITU-T[1] et l'ISO[2] qui, par

[1] *International Telecoms Union*.
[2] *International Organization for Standardization*.

l'intermédiaire de la spécification *Fast Infoset* [FI], proposent de basculer de XML vers un format binaire afin de réduire de façon significative le flux des données. Visant spécifiquement les services Web, le W3C a publié 3 recommandations[1] censées optimiser la transmission et la manipulation de données binaires dans un message SOAP. Ces travaux sont encore à l'origine de nombreuses discussions, notamment concernant le problème de l'interopérabilité. En effet, pas moins d'une douzaine de solutions additionnelles à XML pour les données binaires sont déjà utilisées par les industriels.

De toute façon, soit par l'institution de nouvelles spécifications, soit par l'optimisation des échanges de messages sur un réseau, les services Web trouveront la bonne voie permettant de surmonter leurs problèmes de performance. Dans cette perspective, le système de notification d'événements actuel de LEICA pourrait devenir un système s'appuyant sur des services Web sans souffrir des problèmes de performance actuellement inhérents à SOAP.

– *La vérification de la politique de collaboration*

Les réseaux de Petri se sont montrés un outil bien adapté pour la modélisation des règles politiques. Outre une représentation graphique qui se rapproche de celle des règles politiques, nous avons pu utiliser une approche compositionnelle pour la construction d'un RdP à partir d'une règle. Rappelons cependant que nous n'avons utilisé les RdPs que pour définir la sémantique de l'exécution des règles, alors qu'il s'agit d'un modèle théorique permettant d'analyser et de vérifier le comportement de systèmes complexes.

Il serait donc possible d'utiliser une modélisation en RdP pour la vérification formelle de la politique de collaboration d'une *SuperSession*. Le but serait de trouver de possibles conflits ou incohérences à l'intérieur d'une règle spécifique, ainsi qu'éventuellement de possibles dépendances entre règles (règles qui ont été originalement traitées indépendamment les unes des autres). Un exemple, relatif au premier cas, serait d'identifier dans une règle des conditions (imposées par des prédicats) qui ne soient jamais satisfaites (règles mortes). Dans le deuxième cas, nous pourrions, par exemple, rechercher des règles qui satisfassent une même condition ou une même notification d'événement, mais qui occasionneraient des actions contradictoires (par exemple, connecter et déconnecter un utilisateur donné).

– *Déploiement dynamique des applications collaboratives clientes et P2P*

Dans l'approche faiblement couplée de LEICA, l'autonomie des applications intégrées est également préservée sur le plan de leur installation dans l'environnement hôte utilisateur. En fait, les clients des applications client/serveur ou multiserveurs et les applications paires des applications P2P doivent être installés sur les postes utilisateurs exactement comme s'ils étaient utilisés hors LEICA.

Nous pourrions néanmoins envisager un environnement capable de configurer sur demande l'environnement hôte utilisateur. Ainsi, lorsqu'un utilisateur déciderait de rejoindre une *SuperSession*, LEICA se chargerait de déployer dynamiquement les applications clientes [2] des applications collaboratives utilisées en support de la *SuperSession*. Outre le fait de faciliter l'accès à des utilisateurs nomades (des utilisateurs changeant couramment d'ordinateur pour accéder à l'environnement), cette approche assurerait aux utilisateurs de toujours disposer de la dernière version de l'application sans

[1] XOP (*XML-binary Optimized Packaging*) [W3C-05a], MTOM (*SOAP Message Transmission Optimization Mechanism*) [W3C-05a], et RRSHB (*Resource Representation SOAP Header Block*) [W3C-05c].

[2] Nous entendons par "applications clientes" (i) les clients des applications collaboratives client/serveur et multi-serveurs et (ii) les applications paires des applications P2P.

forcement passer par des étapes complexes de téléchargement, d'installation et de configuration. De la même façon, ces étapes ne seraient plus nécessaires lorsque de nouvelles applications seraient intégrées à LEICA.

Une solution possible consisterait à déployer les applications à partir du Web en utilisant des mécanismes tels que *Java Web Start* de *Sun* [JWS] ou *No-Touch Deployment* de *Microsoft* [MSDN-02]. *Java Web Start* est une solution de déploiement à partir du Web pour les applications Java. Il permet l'installation d'une application *stand-alone* grâce à un simple clic dans un navigateur. Le grand avantage de *Java Web Start* est qu'il n'est pas nécessaire de modifier une application Java pour qu'elle puisse être déployée avec cette technologie. Le problème est qu'une telle solution n'est applicable que pour les applications développées avec la plate-forme Java. Une contrainte similaire est imposée par *Microsoft* et sa solution *No-Touch Deployment*, qui n'est utilisable que sur la plate-forme *Windows*.

– *Le passage à l'échelle*

Une fois que nous disposerons d'une implémentation complète de LEICA, il sera envisageable de réaliser des tests pour analyser le comportement de l'environnement lors du passage à l'échelle. D'un côté, différentes expérimentations devront être réalisées pour évaluer le temps de réponse de LEICA en fonction des performances du système de notification d'événements et pour vérifier notamment l'impact de nombreuses applications P2P dans une même *SuperSession*. Nous devrons également vérifier le temps moyen pris par le module d'exécution des règles politiques pour exécuter l'ensemble des règles : la performance devra être testée non seulement en fonction du nombre de règles, mais également en fonction de la complexité de ces règles (en prenant en compte des règles composées de plusieurs widgets *Latest* et *Earliest*).

Annexe A

Le "fichier de données spécifiques" et le "fichier d'attributs et d'API"

A.1. Le format du fichier de données spécifiques

Le fichier de données spécifiques est un *XML Schema*, comme nous avons illustré dans la Figure A.1. Ce fichier définit en fait une partie de la structure du document XML[1] qui sera repassé à l'application collaborative lorsque LEICA lui demandera de faire partie d'une *SuperSession*. Cette partie correspond en fait à l'élément <sSspecs>.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace=NAMESPACE  // Espace de noms de ce fichier
            xmlns= NAMESPACE           // Espace de noms par défaut
            elementFormDefault="qualified">
   <xsd:element name="sSspecs">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ... />        // Définition des paramètres
            ...
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Figure A.1 Fichier de données spécifiques

```
<xsd:element name="Param1" type="xsd:integer"/> // Valeur simple
-------------------------------------------------------------------------------

<xsd:element name="Param2">      // Ensemble de valeurs
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="element1" type="xsd:string"/>
         <xsd:element name="element2" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Figure A.2 Deux exemples de spécification de paramètres

[1] Comme nous l'indiquerons dans l'Annexe B (section B.2).

<sSspecs> est un élément complexe représentant une liste d'éléments, où chaque élément doit correspondre à un paramètre spécifique dont l'application a besoin pour créer une session collaborative conventionnelle (que nous appelons de session spécifique).

Pour spécifier un paramètre il faut tout simplement suivre le format *XML Schema*. Dans la Figure A.2 nous illustrons deux exemples de spécification de paramètres.

A.2. Le format du fichier d'attributs et d'API

Le fichier d'attributs et d'API est un fichier XML ayant pour balise racine <CA>, comme nous le voyons dans la Figure A.3. <CA> possède trois attributs : "CAid" représente l'identificateur unique de l'application ; "wsRouter" indique l'URL du *RPC router* sur les services Web du Wrapper doivent être déployés ; et "uddiPubURL" indique l'URL de publication du registre UDDI privé de LEICA. En plus, il contient deux éléments : <attributes> et <API>.

```
<CA CAid="..." wsRouter="..." uddiPubURL="...">
   <attributes>
      <name> string </name>
      <description> string </description>
      <type> string </type>
      <maxRolePerUser> nonNegativeInteger </maxRolePerUser>
      <distribution> APP_DISTR </distribution>
      <userApplication>
         <webCallPattern url="http://...">
            <param name="string" value=PARAM_VALUE />
            ...
            <param name="string" value=PARAM_VALUE />
         </webCallPattern>
         ---------------------- OU -----------------------
         <appCallPattern system=OS_TYPE call=PATH>
            <modifier name="string" value=PARAM_VALUE />
            ...
            <modifier name="string" value=PARAM_VALUE />
         </appCallPattern>
      </userApplication>
   </attributes>
   <API>
     <evtsAPI>   ...   </evtsAPI>
     <actsAPI>   ...   </actsAPI>
     <gestionAPI>   ...   </gestionAPI>
   </API>
</CA>
```

```
APP_DIST= "client-server" | "multi-servers" | "p2p"
PARAM_VALUE=  "model:Uid" |          //identificateur de l'utilisateur
              "model:Uid.attName"|   //remplacer "attName" avec un attribut d'utilisateur
              "string" |             // une valeur constante
              "action%paramName" |   //remplacer "paramName" avec un paramètre de l'action
                                     CONNECT de l'API de gestion
OS_TYPE = "unix"|"windows"
PATH =   "java className" |          // remplacer "className" par le nom de la classe
         "string"                    // le chemin complet/relatif plus le nom du fichier
                                     exécutable de l'application
```

Figure A.3 Le fichier d'attributs et d'API[1]

[1] Les balises et les valeurs en gris indiquent qu'elles ne sont pas encore supportées par le prototype actuel.

A.2.1. L'élément <attributes>

Cette balise contient des éléments qui correspondent à des attributs caractérisant l'application collaborative (illustrés dans la Figure A.3) :

- <name> – le nom de l'application ;
- <description> – une description informelle de l'application ;
- <type> – le type d'application collaborative ;
- <maxRolePerUser> – une valeur entière où "0" indique que cette application ne support pas la notion de rôle, et "*n*" (*n*>0) indique qu'elle support cette notion et que chaque utilisateur ne peut être associé qu'à *n* rôles simultanément ;
- <distribution> – l'architecture de distribution de l'application ;
- <userApplication> – les informations nécessaires pour exécuter l'application cliente[1] ; en fonction du type de l'application cliente, cet élément peut contenir soit un élément <webCallPattern> soit une liste d'éléments <appCallPattern> (un pour chaque type de système) :
  - <webCallPattern> – spécifié pour l'application collaborative dont le client est une application Web ; il présent un attribut ("url"), utilisé pour spécifier l'URL d'accès à l'application, et une liste d'éléments <param> spécifiant les paramètres qui doivent être concaténés à la fin de cette l'URL ;
  - <appCallPattern> – spécifié pour l'application collaborative dont le client est une application *stand-alone* ; il présent l'attribut "system", indiquant le type de système, et l'attribut "call", utilisé pour spécifier le chemin et le nom du fichier exécutable (ou le nom de la classe Java) ; il contient également une liste d'éléments <modifiers> spécifiant les modificateurs dont l'appel au fichier exécutable (ou à la classe Java) doit être suivi.

A.2.2. L'élément <API>

Comme il est illustré dans Figure A.4, Cette balise contient des éléments permettant de spécifier l'API événements/actions (à travers les éléments <evtsAPI> et <actsAPI>) et l'API de gestion (à travers l'élément <gestionAPI>) de l'application collaborative.

A.2.2.1. Les éléments <evtsAPI> et <actsAPI>

Dans l'élément <evtsAPI>, une liste d'éléments <event> est définie pour spécifier les événements qui l'application est capable de notifier. Dans l'élément <actsAPI>, une liste d'éléments <action> est définie pour spécifier les actions que l'application est capable de traiter.

Chaque <event> et chaque <action> contiennent un attribut "type" utilisé pour spécifier le type de l'événement ou de l'action respectivement[2]. Ces éléments contiennent également une liste d'éléments <param> spécifiant les paramètres véhiculés par une notification d'événement ou par une requête d'exécution d'action.

---

[1] Nous entendons par "application cliente" (i) le clients d'une application collaborative client/serveur ou multiserveurs ou (ii) l'application paire d'une application P2P.

[2] Le type d'un événement (ou d'une action) doit être unique dans l'ensemble d'événements (ou d'actions) d'une application donnée.

```
<API>
   <evtsAPI>
      <event type="string">
         <param> ... </param>
         ...
      </event>
      ...
   </evtsAPI>
   <actsAPI>
      <action type="string">
         <param> ... </param>
         ...
      </action>
      ...
   </actsAPI>
   <gestionAPI>
   ...
   </gestionAPI>
</API>
```

Figure A.4 L'élément <API>

```
<param name="string" valueType=VALUE_TYPE/>   // paramètre simple
-------------------------------------------------------------------------------
<param name="string">     // paramètre simple avec des valeurs prédéfinis
   <restriction base=VALUE_TYPE>
      <enumeration value="string"/>
      ...
   </restriction>
</param>
-------------------------------------------------------------------------------
<param name="string">      // paramètre correspondant à un vecteur de valeurs
   <sequence base=VALUE_TYPE size=SIZE/>
</param>
```

```
VALUE_TYPE=    NUMERIC_TYPE | "string" | "boolean" | | "duration"| "dateTime" |
               "time"| "date" | "gYearMonth" | "gYear" | "gMonthDay"| "gDay" |
               "gMonth" | "hexBinary" | "base64Binary" | "anyURI" |
               "QName" | "NOTATION" // tous les Built-in Datatypes de XML Schema
NUMERIC_TYPE= "decimal" | "float" | "double" | "integer" |
               "nonPositiveInteger" | "negativeInteger" | "nonNegativeInteger" |
               "positiveInteger" | "int" | "short" | "byte" | "long"
SIZE=  "n" |  // nombre entier
       "*"    // taille pas définie
```

Figure A.5 Les types possibles d'éléments <param>

Comme illustré dans la Figure A.5, un paramètre peut représenter une valeur simple ou un vecteur de valeurs. Dans le premier cas, on peut restreindre l'ensemble de valeurs.

A.2.2.2. L'élément <gestionAPI>

La finalité de cet élément est de permettre de spécifier les actions de l'API de gestion que l'application n'est pas capable de traiter. Ainsi, <gestionAPI> peut contenir des éléments <action> dont les attributs "type" peuvent avoir les valeurs CONNECT, DISCONNECT et RESOURCE. Les deux premiers types d'actions étant obligatoire, ils doivent apparaître soit dans l'élément <actsAPI> soit dans <gestionAPI>, dans le deuxième cas spécifiant justement que l'application n'est pas capable de traiter la requête d'exécution d'action (elle sera en fait traitée par le *LClient*).

A.2.3. Les événements et les actions de l'API de gestion

La Figure A.6 présente les trois types d'événements de gestion qu'une application collaborative (les applications non collaboratives ne sont pas concernées) doit être capable de notifier. Ces types d'événements sont prédéfinis et donc ne doivent pas apparaître dans le fichier d'attributs et d'API.

```
<event type="CONNECT">
   <param name="spSid" valueType="string"/>
   <param name="Uid" valueType="string"/>
   <param name="sRlid" valueType="string"/>
</event>
<event type="DISCONNECT">
   <param name="spSid" valueType="string"/>
   <param name="Uid" valueType="string"/>
</event>
<event type="RESOURCE">   // Optionnel
   <param name="spSid" valueType="string"/>
   <param name="uri" valueType="anyURI"/>
   <param name="type" valueType="string"/>
   <param name="access">
      <restriction base="string">
         <enumeration value="READ"/>
         <enumeration value="WRITE"/>
         <enumeration value="READ/WRITE"/>
      </restriction>
   </param>
</event>
```

Figure A.6 Les événements de l'API de gestion concernant les applications collaboratives

Comme il est illustré dans la Figure A.7, les action des l'API de gestion pour les applications collaboratives correspondent aux mêmes types des événements : CONNECT, DISCONNECT et RESOURCE. Le format des actions CONNECT et DISCONNECT reste ouvert. La seule contrainte est qu'ils doivent présenter le paramètre "Uid" pour identifier l'utilisateur[1].

Une deuxième partie de l'API de gestion concerne les événements et les actions de gestion relatifs à LEICA. Dans la Figure A.8 nous détaillons la spécification de ces événements et actions.

[1] Rappelons que le paramètre "spSid" est défini par défaut pour tout événement et toute action.

```
<action type="CONNECT" >
   <param name="spSid" valueType="string"/>
   <param name="Uid" valueType="string"/>
   ...
</action>
<action type="DISCONNECT">
   <param name="spSid" valueType="string"/>
   <param name="Uid" valueType="string"/>
   ...
</action>
<action type="RESOURCE">
   <param name="spSid" valueType="string"/>
   <param name="uri" valueType="anyURI"/>
   <param name="access">
      <restriction base="string">
         <enumeration value="READ"/>
         <enumeration value="WRITE"/>
         <enumeration value="READ/WRITE"/>
      </restriction>
   </param>
</action>
```

Figure A.7 Les actions de l'API de gestion concernant les applications collaboratives

```
<event type="CONNECT">
   <param name="Uid" valueType="string"/>
   <param name="name" valueType="string"/>
   <param name="Rlid" valueType="string"/>
   <param name="ip" valueType="string"/>
</event>
<event type="DISCONNECT">
   <param name="Uid" valueType="string"/>
</event>
--------------------------------------------------------------------------------
<action type="DISCONNECT">
   <param name="Uid" valueType="string"/>
</action>
```

Figure A.8 L'API de gestion concernant LEICA.

Annexe B

Le "fichier de configuration de la SuperSession"

Dans la Figure B.1 nous illustrons le format XML du fichier de configuration de la *SuperSession*. La racine <SuperSession> présent l'attribut "id" correspondant à l'identificateur unique de la *SuperSession*, ainsi que trois éléments : <GSMInfo>, <IAInfo> et <rules>.

```
<SuperSession id="string" >
   <GSMInfo>
      <context>
         <name> string </name>
         <description> string </description>
      </context>
      <scheduling>
         <starting> START_TIME </starting>
         <duration> END_TIME </duration>
      </scheduling>
      <roles>
         <role id="string">
            <description> string </description>
            <membership>
            ...
            </membership>
            <maxSimUsers> integer </maxSimUsers>
            <rights> ASSIGN </rights>
         </role>
         ...
      </roles>
   </GSMInfo>
   <IAInfo>
      <CA id="string">   ...   </CA>
      ...
   </IAInfo>
   <rules>
      <rule>   ...   </rule>
      ...
   </rules>
</SuperSession>
```

```
START_TIME= CCYY-MM-DDThh:mm:ss | hh:mm:ss-[Su][Mo][Tu][We][Th][Fr][Sa] | null
END_TIME =  CCYY-MM-DDThh:mm:ss | hh:mm:ss | PyYmMdDThHmMsS | null
ASSIGN=     all | n first | n last | null  // n>0
```

Figure B.1 Format du fichier de configuration de la SuperSession

B.1. L'élément <GSMInfo>

<GSMinfo> regroupe les éléments suivants (Figure B.1) :

- <context> – permet de définir un nom (à travers l'élément <name>) et une description (à travers l'élément <description>) pour la *SuperSession* ;
- <scheduling> – permet de programmer le début et la fin de la *SuperSession* ;
- <roles> – permet de définir une liste de rôles généraux où chaque élément <role> correspond à un rôle ; l'élément <role> contient, en plus de l'attribut "id", les éléments :
  - <description> – permet de faire une description informelle du rôle ;
  - <membership> – permet de définir comment un rôle est associé à un utilisateur ; si c'est de manière statique, on prédéfinit une liste avec la composition du groupe (en spécifiant une liste d'éléments <user>, comme illustré en haut de la Figure B.2) ; si c'est à l'utilisateur de choisir, on peut juste définir un mot de passe (en spécifiant un élément <password>, comme illustré en bas de la Figure B.2), ou simplement omettre l'élément <membership> en indiquant qu'il s'agit d'une *SuperSession* ouverte où n'importe quel utilisateur peut se connecter en choisissant n'importe quel rôle généraux parmi ceux définis [1];
  - <maxSimUsers> – permet de définir le nombre maximum d'utilisateurs autorisés pour ce rôle (la valeur *-1*, ou l'absence de cet élément, indique qu'il n'y a pas de limite) ;
  - <rights> – permet de donner au rôle des droits d'administration et de définir parmi les utilisateurs associés à ce rôle ceux qui auront ces droits (la valeur *null*, ou l'absence de cet élément, indique que le rôle ne possède pas de droits d'administration).

```
        <membership>
           <user id="string"  password="string" />
           ...
        </membership>
-------------------------------------------------------------------------------
        <membership>
           <password> string </password>
        </membership>
```

Figure B.2 Les possibles spécifications de l'élément <membership>

B.2. L'élément <IAInfo>

Le <IAInfo> permet de définir et configurer les applications utilisées dans la *SuperSession*. Ainsi, il contient une liste d'éléments <CA>, chacun correspondant à une application collaborative (ou éventuellement, non collaborative).

Comme nous illustrons dans la Figure B.3, chaque élément <CA>[2] présent l'attribut "id", identifiant l'application, et contient un élément <attributes>, ainsi qu'une liste d'éléments <spS>. L'élément <attributes> est en fait une simple copie de l'élément portant le même nom défini dans le fichier d'attributs et d'API de cette application (illustré précédemment dans la Figure A.3).

[1] Soit tous les éléments <role> contiennent un élément <membership>, soit aucun élément <role> ne contient de <membership>.
[2] Dans le cas des applications non collaboratives, cet élément contient seulement l'attribut "id".

```
<CA id="string">
      <attributes>
      ...
      </attributes>
      <spS id="string">
         <scheduling>
            <starting> START_TIME </starting>
            <duration> END_TIME </duration>
         </scheduling>
         <roles type=string>
         ...
         </roles>
         <spSspecs>
         ...
         </spSspecs>
      </spS>
      ...
   </CA>
```

```
START_TIME= CCYY-MM-DDThh:mm:ss |      // programmée comme la SuperSession
            hh:mm:ss-[Su][Mo][Tu][We][Th][Fr][Sa] | //programmée
            null
END_TIME =  CCYY-MM-DDThh:mm:ss |      // programmée
            hh:mm:ss |    // programmée
            PyYmMdDThHmMsS |    // programmée
            null           //arrêtée manuellement ou suite aux règles politiques
```

Figure B.3 Détail sur l'élément <CA>

Chaque élément <spS> permet de spécifier une session spécifique pour l'application collaborative. Toujours dans la Figure B.3, nous voyons que cet élément, en plus de l'attribut "id" (spécifiant l'identificateur de la session spécifique), contient trois éléments :

- <scheduling> – permet de programmer le début et la fin de la session spécifique pour la *SuperSession* ;
- <roles> – permet de définir une liste de rôles spécifiques (à travers la spécification d'une liste d'éléments <role>) ; en fonction de la valeur de l'attribut "type" nous avons différentes spécifications (comme illustré dans Figure B.4) :
  - "type= "none"" – l'application ne support pas la notion de rôle ; il est possible de définir la liste d'utilisateur qui peuvent se connecter à la session spécifique, ou juste un mot de passe ; [1]
  - "type="static"" [2] – on définit pour chaque rôle une liste d'utilisateurs et les mots de passes associés ;
  - "type="automatic"" – on définit pour chaque rôle une liste de rôles généraux, pour faire une association automatique ;
  - "type="anonymous"" – c'est à l'utilisateur de choisir un rôle spécifique ;
- <spSspecs> – le format de cet élément est définit dans le fichier de données spécifiques de l'application collaborative (présenté précédemment dans la section 0).

[1] Dans tous les cas, si l'élément <membership> n'est pas spécifié, nous avons une session ouverte.
[2] Pour avoir une association statique de rôles spécifiques il faut que les rôles généraux soient également associés aux utilisateurs de manière statique.

```
<roles type="none">
  <membership>   // élément optionnel
    <user id="string" password="string"/> //liste statique d'utilisateurs, session fermée
    ...
    ------------------------------------------------ OU ------------------------------------------------
    <password> string </password>  // session fermée
  </membership>
</roles>
-------------------------------------------------------------------------------
<roles type="static">
  <role id="string">
    <description> string </description>
    <membership>
      <user id="string"  password="string" /> //liste statique d'utilisateurs par rôle
      ...
    </membership>
  </role>
  ...
</roles>
-------------------------------------------------------------------------------
<roles type="automatic">
  <role id="string">
    <description> string </description>
    <membership>
      <password> string </password>
    </membership>
    <baseRole> string </baseRole> // chaque baseRole correspond à un rôle général
    ...
  </role>
</roles>
-------------------------------------------------------------------------------
<roles type="anonymous">
  <role id="string">
    <description> string </description>
    <membership>
      <password> string </password>
    </membership>
  </role>
</roles>
```

Figure B.4 Les quatre spécifications possibles de l'élément <roles>

B.3. L'élément <rules>

L'élément <rules> permet de spécifier la politique de collaboration de la *SuperSession*. Il contient une liste d'éléments <rule> (illustré précédemment dans la Figure B.1), chacun correspondant à une règle politique.

Comme il est illustré dans la Figure B.5, chaque élément <rule> contient deux éléments. Le premier pouvant être un <event>, un <trigger>, un <earliest>, ou un <latest> [1], et le deuxième pouvant être un <final-predicate> ou un <actions>.

[1] Définissons ELEMENT comme étant un de ces éléments (ELEMENT = <event> | <trigger> | <earliest> | <latest>).

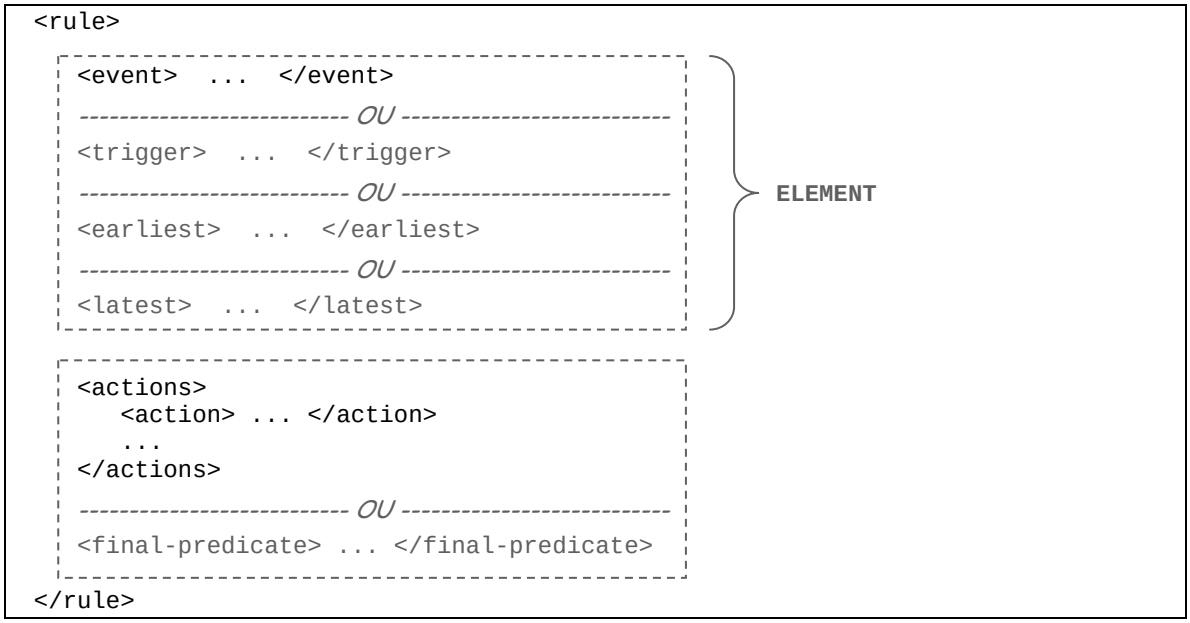

Figure B.5 L'élément <rule>

Tandis que <event> et <trigger> représentent des éléments simples, <earliest> et <latest> sont des éléments permettant de regrouper d'autres éléments. Comme nous pouvons remarquer dans la Figure B.6 et dans la Figure B.7, cela est possible à travers l'élément <group>[1]. Remarquons également que chaque <earliest> ou chaque <latest> peut contenir un élément <predicate>. En fait, ce <predicate> est spécifié lorsque nous avons un *Predicate* couplé à un *Latest* ou à un *Earliest*, à l'exception de la situation suivante : que le *Predicate* soit placé dans la règle juste avant le(s) *Action*(s). Dans ce cas, le *Predicate* sera représenté par un élément <final-predicate>, que nous voyons en bas de la Figure B.5.

```
<earliest>
   <group>
      ELEMENT
      ...
   </group>
   <predicate> ... </predicate> // Optionnel
</earliest>
```

Figure B.6 L'élément <earliest>

```
<latest tw="integer">
   <group>
      ELEMENT
      ...
   </group>
   <predicate> ... </predicate> // Optionnel
</latest>
```

Figure B.7 L'élément <latest>

Dans la Figure B.8 nous détaillons l'élément <final-predicate>. Remarquons qu'en plus de l'élément <predicate>, il contient en élément <true> et, optionnellement, un

[1] En fait les formats des éléments "earliest" et "latest" sont identiques, à l'exception de l'attribut "tw" présent dans le "latest" (permettant de spécifier le "temps d'attente").

élément <false>, qui permettent en fait de spécifier la liste d'actions à exécuter lorsque le prédicat est évalué à vrai ou à faux respectivement.

```
<final-predicate>
   <predicate>  ...  </predicate>
   <true>
      <actions>  ...  </actions>
   </true>
   <false>    // Optionnel
      <actions>  ...  </actions>
   </false>
</final-predicate>
```

Figure B.8 L'élément <final-predicate>

B.3.1. L'élément <event>

L'élément <event> correspond en fait à un *widget Event* dans une règle. Ainsi, comme nous illustrons dans la Figure B.9, il présent les mêmes attributs "from" et "type" du *widget*, en plus de l'attribut "id", ce dernier correspondant à l'identificateur numérique associé au *widget*. En plus, il contient une liste d'éléments <param>, dans le quel on peut spécifier des motifs, ou *matching patterns* (montrés en bas de la Figure B.9), appliqués à la valeur (ou aux valeurs) du paramètre respectif.

Comme dans le cas du <earliest> et du <latest>, un <event> peut également contenir un élément <predicate>, indicant que le *widget Event* est couplé à un *Predicate*. De la même façon, l'exception se fait lorsque le *Predicate* est placé dans la règle juste avant le(s) *Action*(s). Il est dans ce cas représenté par un élément <final-predicate>.

```
<event from="string" type="string" id="integer">
   <param name="string"  >
      <value>
        VALUE_TYPE
        --------- OU ---------
        STRING_PATTERN
        --------- OU ---------
        NUMBER_RANGE
      </value>
      ...
   </param>
   ...
   <predicate> ... </predicate>
</event>
```

```
VALUE_TYPE = ... // Définie dans la Figure A.5
STRING_PATTERN = +["*"|"?"|string]    // * correspond à toute string
                                      // ? correspond à un caractère

NUMBER_RANGE= OPERATOR NUMERIC_TYPE // Définie dans la Figure A.5
OPERATOR=     ">" | ">=" | "<" | "<=" | "!="
```

Figure B.9 L'élément <event>

B.3.2. Les éléments <trigger> et <predicate>

L'élément <trigger> correspond à un *widget Trigger* dans une règle. Rappelons que le *Trigger* peut définir une condition (à travers un prédicat) portant sur des composantes de l'état global de la *SuperSession*. Nous représentons dans la Figure B.10 le format de ce prédicat. Dans la Figure B.11 nous avons en détail la syntaxe définie pour les références à des composantes de l'état global de la *SuperSession*.

```
<trigger>   TRIGGER_PR   </trigger>

TRIGGER_PR=  VALUE BINARY_OP VALUE |
             *[[UNARY_OP](TRIGGER_EXP) CONNECTOR] [UNARY_OP](TRIGGER_EXP)
VALUE=       MODEL_REF |
             VALUE_TYPE // Définie dans la Figure A.5
UNARY_OP=    "!"
BINARY_OP=   ">" | "<" | ">=" | "<=" | "==" | "!="
CONNECTOR=   "&&" | "||"
MODEL_REF=   ... // Définie dans la Figure B.11
```

Figure B.10 Syntaxe d'un prédicat spécifié dans un élément <trigger>

MODEL_REF = **ELEMENT_REF** | **ATTR_REF** | **SET_REF**

Type	*Value*	*Description / Exemples*
ELEMENT_REF	**PARENT_NAME**("*id*") PARENT_NAME = "CA" | "NA" | "Rl" | "U" | "Rs"	Références à des éléments (composantes de l'état global) portant un identificateur unique dans la *SuperSession.* Ex : CA("Chat") , U("rgomes")
	PARENT_NAME("*id*")+[.**CHILD_NAME**("*id*")] CHILD_NAME = "spSi" | "sRl" | "Mb" | "pUid"	Références aux autres éléments portant un identificateur. Ex : CA("Chat").spS("ses1") , CA("Chat").spS("ses1").sRl("Prof")
ATTR_REF	ELEMENT_REF.*attributeName*	Références aux attributs de chaque élément définit dans la *SuperSession.* Ex : U("rgomes").ipAddress , CA("Chat").spS("ses1").startingTime
SET_REF	PARENT_NAME.size | PARENT_NAME("*id*")*[.CHILD_NAME("*id*")] CHILD_NAME.size	Référence à l'attribut *size* spécifique à des ensembles. Ex : U.size , CA("Chat").spS.size
	PARENT_NAME.HAS_CALL | PARENT_NAME("*id*")*[.CHILD_NAME("*id*")] CHILD_NAME.HAS_CALL HAS_CALL = has("*id*") | has(*attributeName*==VALUE)| has(CHILD_NAME==VALUE)	Référence à la méthode *has* spécifique à des ensembles, qui permet de vérifier la présence d'un élément dans un ensemble donné. Ex : U.has("rgomes") , U.has((URlid =="Moniteur")

Figure B.11 Syntaxe des références à des composantes de l'état global de la SuperSession

Quant au <predicate>, il correspond à un *widget Predicate* dans une règle. Comme nous illustrons dans la Figure B.12, la syntaxe du prédicat est similaire à celle définie pour

le <trigger>, sauf qu'en plus des composantes de l'état global de la *SuperSession*, le prédicat peut également porter sur des paramètres des *widgets Event* présents dans la même règle.

```
<predicate>  PREDICATE  </predicate>

PREDICATE=   VALUE_PR BINARY_OP VALUE_PR |
             *[[UNARY_OP](PREDICATE) CONNECTOR] [UNARY_OP](PREDICATE)
VALUE_PR=    VALUE |
             PARAM_REF
UNARY_OP=    ... // Définie dans la Figure B.10
BINARY_OP=   ... // Définie dans la Figure B.10
CONNECTOR=   ... // Définie dans la Figure B.10
VALUE=       ... // Définie dans la Figure B.10
PARAM_REF=   %ID.PARAM_NAME  |
             %ID.PARAM_NAME[INDEX]
ID=          integer  // l'identificateur numérique d'un Event présent dans la même règle
PARAM_NAME=  string  // le nom d'un paramètre d'un Event
INDEX=       integer  // la position dans le vecteur de valeurs du paramètre
```

Figure B.12 Syntaxe d'un prédicat spécifié dans un élément <predicate>

B.3.3. L'élément <action>

Le dernier élément dans une règle est le <action>, qui correspond au *widget Action*. En plus des paramètres "to" et "type", il contient une liste d'éléments <param>, dans le quel on peut spécifier des valeurs. Comme illustré dans la Figure B.13, ces valeurs peuvent être des constantes, des références à des composantes de l'état global de la *SuperSession*, ou des références à des paramètres provenant des notifications d'événement.

```
<action to="string" type="string" >
   <param name="string"  >
      <value>
        VALUE_TYPE
        --------- OU ---------
        MODEL_REF
        --------- OU ---------
        PARAM_REF
      </value>
      ...
   </param>
   ...
</action>

VALUE_TYPE = ... // Valeur constante, définie dans la Figure A.5
MODEL_REF = ...  // Référence à une composante d'état, définie dans la Figure B.11
PARAM_REF = ...  // Référence à un paramètre provenant d'un Event, définie dans la Figure B.12
```

Figure B.13 L'élément <action>

Références bibliographiques

Publications de l'auteur

[Hoyos-06] Hoyos-Rivera, G.J., Gomes, R.L. Willrich, R., Courtiat, J.P.
"CoLab: Co-Navigation sur le Web"
NOuvelles TEchnologies de la REpartition (NOTERE'06)
Toulouse (France), Jun. 2006.

[Gomes-05d] Gomes, R.L., Hoyos-Rivera, G.J., Courtiat, J.P.
"LEICA: Un environnement faiblement couplé pour l'intégration d'applications coopératives"
NOuvelles TEchnologies de la REpartition (NOTERE'05)
pp.189-195, Québec (Canada), Aug. 2005.
ISBN 2-9809043-0-9

[Gomes-05c] Gomes, R.L., Hoyos-Rivera, G.J., Courtiat, J.P.
"LEICA: Loosely-coupled Environment for Integrating Collaborative Applications"
16th International Workshop on Database and Expert System Applications (DEXA 2005) - *Web Based Collaboration workshop* (WBC 2005), IEEE Computer Society Press
pp.635-639, Copenhagen (Denmark), Aug. 2005
ISBN 0-7695-2424-9

[Gomes-05b] Gomes, R.L., Hoyos-Rivera, G.J., Courtiat, J.P.
"Integrating Collaborative Applications with LEICA"
3rd IEEE International Conference on Information Technology: Research and Education (ITRE 2005), IEEE Computer Society Press
pp.292-296, Hsinchu (Taiwan), Jun. 2005
ISBN 0-7803-8932-8

[Gomes-05a] Gomes, R.L., Hoyos-Rivera, G.J., Courtiat, J.P.
"Loosely-Coupled Integration of CSCW Systems"
The 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), LNCS 3543, Springer
pp.38-49, Athens (Greece), Jun. 2005
ISBN 3-540-26262-8

[Hoyos-05b] Hoyos-Rivera, G.J., Gomes, R.L., Courtiat, J.P.
"CoLab: A Flexible Collaborative Web Browsing Tool"
19th IEEE International Conference on Advanced Information Networking and Applications (AINA'05), IEEE Computer Society Press
v.1, pp.501-506, Taipei (Taiwan), Mar. 2005
ISBN 0-7695-2249-1

[Hoyos-05a] Hoyos-Rivera, G.J., Gomes, R.L., Courtiat, J.P., Wilrich R.
"Collaborative Web Browsing Tool Supporting Audio/Video Interactive Presentations"
14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE'2005), IEEE Computer Society Press
pp.78-83, Linköping (Sweden), Jun. 2005
ISBN 0-7695-2362-5

[Gomes-03c] Gomes, R.L., Hoyos-Rivera, G.J., Courtiat, J.P.
"Regarding the Integration of Collaborative Applications Into Virtual Worlds."
Cooperative Information Systems International Conference (CoopIS'03), LNCS 2889, Springer, *Short paper*
pp.16-17, Catanie (Italie), 3-7 Nov. 2003
ISBN: 3-540-20494-6

[Gomes-03b] Gomes, R.L.
" Conception et Developpement d'environnements virtuels collaboratifs"
4ème Congrès des Doctorants de l'Ecole Doctorale Systèmes
Toulouse (France), Oct. 2003.

[Gomes-03a] Gomes, R.L., Hoyos-Rivera, G.J., Courtiat, J.P.
"Collaborative Virtual Environments: Going Beyond Virtual Reality"
IEEE International Conference on Multimedia (ICME'2003), IEEE Computer Society Press
v.2, pp.105-108, Baltimore (USA), Jul. 2003
ISBN 0-7803-7965-9

[Hoyos-03] Hoyos-Rivera, G.J., Gomes, R.L., Courtiat, J.P., Benabbou, R.
"The Web as Tool for Collaborative e-Learning: the Case of CoLab"
3rd IEEE International Conference on Advanced Learning Technologies (ICALT'03), IEEE Computer Society Press, *Short paper*
pp.312-313, Athens (Greece), Jul. 2003
ISBN 0-7695-1967-9

[Baudin-03] Baudin, V., Courtiat, J.P., Gomes, R.L.; Hoyos-Rivera, G.J., Villemur, T.
"An e-Learning Collaborative Platform for Laboratory Education"
4th IEEE International Conference on Information Technology Based Higher Education and Training (ITHET'03), IEEE Computer Society Press
pp.200-205, Marrakech (Maroc), Jul. 2003

[Hoyos-02] Hoyos-Rivera, G.J., Gomes, R.L., Courtiat, J.P.
"A Flexible Architecture for Collaborative Browsing"
11th IEEE International Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises (WETICE'2002), IEEE Computer Society Press
pp.164-169, Pittsburgh (USA), Jun. 2002
ISBN 0-7695-1748-X

Bibliographie

[Abdel-Wahab-91] Abdel-Wahab, H., Feit, M.
"XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration"
IEEE Conference on Communications Software: Communications for Distributed Applications and Systems (IEEE Tricomm 91), IEEE Computer Society Press
pp. 159-167, Apr. 1991
ISBN 0-87942-649-7

[Ahuja-90] Ahuja, S.R., Ensor, J.R., Lucco, S.E.
"A comparison of application sharing mechanisms in real-time desktop conferencing systems"
ACM Conference on Supporting Group Work, Proceedings of the conference on Office Information Systems
pp.238–248, Cambridge (United States), 1990.

[Alonso-04a] Alonso, G., Casati, F., Kuno, H., Machiraju, V.
"Enterprise Application Integration"
In: *Web Services – Concepts, Architectures and Applications*, Springer-Verlag
pp.67-92, 2004.
ISBN: 3-540-44008-9

[Alonso-04b] Alonso, G., Casati, F., Kuno, H., Machiraju, V.
"Web Services"
In: *Web Services – Concepts, Architectures and Applications*, Springer-Verlag
pp.123-149, 2004.
ISBN: 3-540-44008-9

[Aslst-02] Aslst, W.v.d., Hee, K.v.
"Workflow management: models, methods, and systems"
MIT Press, Cambridge (USA), 2002.

[Barthelmess-05] Barthelmess, P., Ellis, C.A.
"The Neem Platform: an Evolvable Framework for Perceptual Collaborative Applications"
Journal of Intelligent Information Systems
v.25, n.2, pp.207-240, Sep. 2005
ISSN 0925-9902

[BEA-04] BEA Systems
"Service Oriented Architecture Solution Acelerator Guide"
Version 1,0, Revision 2.12,
http://ftpna2.bea.com/pub/downloads/SOA_SAG.pdf

[Begole-99] Begole, J., Rosson, M.B., Shaffer., C.A.
"Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems"
ACM Transactions on Computer-Human Interaction
v.6, n.2, pp.95–132, Jun. 1999.

[Bernard-04] Bernard, S.
"Spécification d'un environnement d'ingénierie collaborative mulitisite - Application à l'industrie aéronautique européenne"
Thèse de doctorat Génie Industriel
École nationale supérieure d'arts et métiers, Centre d'Aix-en-Provence
France, Nov. 2004.

[Berthomieu-91] Berthomieu, B., Diaz, M.
"Modeling and verication of time dependent systems using time Petri nets"
IEEE Transactions on Software Engineering, IEEE Computer Society Press
v.17, n.3, pp.259-273, 1991

[Björkander-03] Björkander, M. & Kobryn, C.
"Architecting Systems with UML 2.0"
IEEE Software, IEEE Computer Society Press
v.20, n.4, Jul.-Aug. 2003

[Bolognesi-87] Bolognesi, T., Brinksma, E.
"Introduction to the ISO specification language LOTOS"
Computer Networks and ISDN Systems, Elsevier Science Publishers
v.14, n.1, pp.25-59, Mar. 1987.

[Bourguin-00] Bourguin, G.
"Un support informatique à l'activité cooperative fondé sur la théorie de l'activité : le projet DARE"
Thèse de doctorat Informatique
Université des sciences et technologies de Lille, Lille
France, Jui. 2000.

[Boyle-02] Boyle, M., Greenberg, S.
"GroupLab Collabrary: A Toolkit for Multimedia Groupware"
Extended Abstracts of the ACM Conference on Computer Support Cooperative Work (CSCW'02), *Workshop on Network Services for Groupware*, ACM press
New Orleans (USA), Nov. 2002.

[Brownsword-04] Brownsword, L.L. et. al.
"Current Perspectives on Interoperability"
Technical Report CMU/SEI-2004-TR-009
Software Engineering Institute, Carnegie Mellon University
51pp., Pittsburgh (USA), Mar. 2004.
http://www.sei.cmu.edu/pub/documents/04.reports/pdf/04tr009.pdf

[Bruckman-97] Bruckman. A.
"MOOSE Crossing: Construction, Community, and Learning in A Networked Virtual World for Kids"
Doctoral Dissertation, MIT Media Lab
Cambridge (USA), 1997

[Capps-00] Capps, M., McGregor, D., Brutzman, D., Zyda, M.
"NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments"
IEEE Computer Graphics and Applications, IEEE Computer Society Press
v.20, n.5, pp.12-15, 2000.
ISSN 0272-1716

[Castro-02] Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.
"Scribe: A large-scale and decentralized application-level multicast infrastructure"
IEEE Journal on Selected Areas in Communications (JSAC), IEEE Computer Society Press
v.20, n.8, pp.1489-1499, 2002
ISSN 0733-8716

[Chiu-02] Chiu, K., Govindaraju, M., Bramley, R.
"Investigating the limits of SOAP performance for scientific computing"
11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), IEEE Computer Society Press
pp.246-254, Edinburgh (Scotland), 2002
ISBN 0-7695-1686-6

[Colyer-03] Colyer, A., Blair, G., Rashid, A.
"Managing Complexity in Middleware"
2nd AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)
pp.21-26, Boston, (USA), Mar. 2003
http://www.cs.ubc.ca/~ycoady/acp4is03/acp4isTR.pdf

[Cosquer-99] Cosquer, F.J.N., Verissimo, P., Krakowiak, S., Decloedt, L.
"Support for Distributed CSCW Applications",
Recent Advances in Distributed Systems, eds. S. Krakowiak and S.K. Shrivastava, LNCS 1752, Springer
pp. 295-326, 2000.

[Courtiat-00] Courtiat, J.P., Santos, C.A.S, Lohr, C., Benaceur, O.
"Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique"
Computer Communications
v.23, n.12, pp.1104-1123, Jul. 2000.
ISSN 0140-3664

[Dalal-78] Dalal, Y.K., Metcalfe, R.M.
"Reverse Path Forwarding of Broadcast Packets"
Communications of the ACM, ACM press
vol.21, n.12, pp.1040-1048, Dec. 1978

[Damianou-01] Damianou, N., Dulay, N., Lupu, E., Sloman, M.
"The Ponder Policy Specification Language"
Workshop on Policies for Distributed Systems and Networks (Policy 2001), LNCS 1995, Springer
pp.18-39, Bristol (UK), Jan. 2001

[Darlagiannis-00] Darlagiannis, V., Georganas, N.D.
"Virtual Collaboration and Media Sharing using COSMOS"
4th WORLD MULTICONFERENCE on Circuits, Systems, Communications & Computers (CSCC'00)
Athens (Greece), Jul. 2000.
http://www.mcrlab.uottawa.ca/papers/314.pdf

[Dewan-99] Dewan, P., Sharma, A.
"An experiment in interoperating heterogeneous collaborative systems"
6th European Conference on Computer Supported Cooperative Work (ECSCW'99), Kluwer Academic Publishers
pp.371-390, Copenhagen (Denmark), 1999
ISBN 0-7923-5947-X

[Diaz-01] Diaz, M.
Les Réseaux de Petri. Modèles Fondamentaux
Traité IC2, Série Informatique et systèmes d'information, Hermes Science Publication
384pp., 2001
ISBN 2-7462-0250-6

[Diaz-03] Diaz, M.
Vérification et mise en œuvre des réseaux de Petri
Traité IC2, Série Informatique et systèmes d'information, Hermes Science Publication
388pp., 2003
ISBN 2-7462-0445-2

[Dix-97] Dix, A.
"Challenges for Cooperative Work on the Web: An analytical approach"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.6, n.2-3, pp.135-156, 1997.
ISSN 0925-9724

[Domingos-97] Domingos, H.J.L. Martins, J.A.L.
"Coordination and Tailorability Issues in the Design of a Generic Large Scale Groupware Platform"
ACM Conference on Supporting Group Work (GROUP'97)*, Workshop on Tailorable Groupware: Issues, Methods, and Architectures*, ACM press
Phoenix (USA), Nov. 1997
ISBN 0-89791-897-5

[Dommel-00] Dommel, H.-P., Garcia-Luna-Aceves, J.J.
"Networking Foundations for Collaborative Computing at Internet Scope"
International ICSC Congress on Intelligent Systems and Applications, Symposium on Interactive and Collaborative Computing (ICC'2000)
Wollongong (Australia), Dec. 2000.
http://www.cse.scu.edu/~hpdommel/publications/peter.isa2k.pdf

[Dommel-99] Dommel, H.P., Garcia-Luna-Aceves, J.J.
"Efficacy of floor control protocols in distributed multimedia collaboration"
Cluster Computing Journal, Springer Science+Business Media B.V.
1573-7543
v.2, n.1, pp.17-33, Mar. 1999
ISSN: 1386-7857

[Dourish-00] Dourish, P., Edwards, W.K.
"A tale of two toolkits: Relating infrastructure and use in flexible CSCW toolkits"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.9, n.1,pp.33-51, 2000
ISSN 0925-9724

[Dourish-92] Dourish, P., Bellotti, V.
"Awareness and coordination in shared workspaces"
ACM Conference on Computer Supported Cooperative Work (CSCW'92), ACM Press
pp.107–114, Toronto (Canada), 1992
ISBN 0-89791-542-9

[Dourish-96] Dourish P.
Open Implementation and Flexibility in CSCW Toolkits
Doctoral Dissertation, University College London
London, 1996

[Dourish-98] Dourish P.
"Using Metalevel Techniques in a Flexible Toolkit for CSCW Applications"
ACM Transactions on Computer-Human Interaction, ACM press
v.5, n.2, pp.109-155, 1998

[Dustdar-04] Dustdar, S., Gall, H., Schmidt, R.
"Web services for Groupware in Distributed and Mobile Collaboration"
12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04)
pp.241-247, Coruna (Spain), 2004
ISBN 0-7695-2083-9

[Edwards-96] Edwards, W.K.
"Policies and Roles in Collaborative Applications"
ACM Conference on Computer Supported Cooperative Work (CSCW'96), ACM press
pp.11-20, Boston (USA), Nov. 1996
ISBN 0-89791-765-0

[Ellis-91] Ellis, C.A., Gibbs, S.J., Rein G.
"Groupware: some issues and experiences"
Communications of the ACM , ACM Press
v.34 , n.1, pp.39-58, 1991.

[Ellis-94] Ellis, C.A., and Wainer, J.
"A Conceptual Model of Groupware"
ACM Conference on Computer Supported Cooperative Work (CSCW'94), ACM press
pp.79-88, Chapel Hill (USA), 1994
ISBN:0-89791-689-1

[Ellis-99] Ellis, C.A.
"Workflow Technology"
Computer-Supported Cooperative Work, Trends in Software (Beaudouin-Lafon eds.)
Pages 29-54, John Wiley & Sons

[Engeström-99] Engeström, Y.
"Activity theory and individual and social transformation"
Perspectives on Activity Theory (Engeström, Miettinen, and Punamäki eds.), Cambridge University Press
pp.19-38, 1999.
ISBN 0-521-43730-X

[Eseryel-02] Eseryel, D., Ganesan, R., Edmonds, G.S.
"Review of Computer-Supported Collaborative Work Systems"
Educational Technology & Society,
v.5, n.2, p.130-136, 2002.
ISSN 1436-4522

[Eugster-03] Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.-M.
"The many faces of publish/subscribe"
ACM Computing Surveys, ACM press
v.35, n.2, pp.114-131, 2003.

[Farias-00a] Farias, C.R.G., Pires, L.F., van Sinderen, M.
"A conceptual model for the development of CSCW systems"
International Conference on the Design of Cooperative Systems (COOP 2000)
pp.189-204, Sophia Antipolis (France), 2000.

[Farias-00b] Farias, C.R.G., Diakov, N.
"Component-Based Groupware Tailorability using Monitoring Facilities"
ACM Conference on Computer Support Cooperative Work (CSCW'00), *Workshop on Component-based Groupware* (CBG2000)
Philadelphia (USA) Dec. 2000
ISBN 90-75176-24-4

[Fox-03] Fox, G., Wu, W., Uyar, A., Bulut, H., Pallickara, S.
"A Web services framework for collaboration and videoconferencing"
Workshop on Advanced Collaborative Environments (WACE'03)
Seattle (USA), Jun. 2003

[Frécon-98] Frécon, E., Nöu, A.A.
"Building distributed virtual environments to support collaborative work"
ACM Symposium on Virtual Reality Software and Technology (VRST'98), ACM Press
pp.105-113, (Taiwan), 1998.
ISBN:1-58113-019-8

[Fuchs-99] Fuchs, L.
"AREA: a cross-application notification service for groupware"
6th European Conference on Computer Supported Cooperative Work (ECSCW'99)
pp.61-80, Copenhagen (Denmark), 1999
ISBN 0-7923-5947-X

[Gangopadhyay-95] Gangopadhyay D., Pree W., Schappert, A.
"Report on the Workshop Framework – Centered Software Development"
ACM Conference on Object Oriented Programming Systems Languages and Applications (Addendum).
pp.100-104, Austin (USA), 1995
ISSN 1055-6400

[Garcia-01] Garcia Lopez, P., Skarmeta, A., Molla, R.R.
"ANTS: A new Collaborative Learning Framework"
European Conference on Computer Supported Collaborative Learning, (Euro CSCL'01)
pp.253-260, Maastricht (Holand), Mar.2001
ISBN 90-5681-097-9

[García-02] García, P., Montalà, O., Pairot, C., Rallo, R., Skarmeta, A.G.
"MOVE: Component Groupware Foundations for Collaborative Virtual Environments"
4th International Conference on Collaborative Virtual Environments (CVE'02), ACM Press
pp.55-62, (Germany), 2002.
ISBN 1-58113-489-4

[Geihs-01] Geihs, K.
"Middleware Challenges Ahead"
IEEE Computer, IEEE Computer Society Press
v.34, n.6, pp.24-31, 2001.
ISSN: 0018-9162

[Genrich-81] Genrich, H.J., Lautenbach, K.
"System modeling with high-level Petri nets"
Theoretical Computer Science,
v.13, n.1, pp.109-136, Jan. 1981.

[Graham-99] Graham, T.C.N., Grundy, J.
"External Requirements of Groupware Development Tools"
Engineering for Human-Computer Interaction, Kluwer Academic Press
pp.363-376, Heraklion (Grèce), 1999

[Green-98] Green, D.
"The Reflection API"
Java™ Tutorial Continued, The: The Rest of the JDK™ (M. Campione, K. Walrath, A. Huml Editeurs), Addison Wesley Professional
pp.699-732, Dec. 1998
ISBN: 0-201-48558-3.
http://java.sun.com/docs/books/tutorial/reflect/

[Greenberg-02] Greenberg, S., Roseman, M.
"Using a Room Metaphor to Ease Transitions in Groupware"
Beyond Knowledge Management: Sharing Expertise (M. Ackerman, V. Pipek, V. Wulf eds.), MIT Press
Revised from report 98/611/02, 2002
http://www.cpsc.ucalgary.ca/grouplab/papers/index.html

[Greenhalgh-00] Greenhalgh, C., Purbrick, J., Snowdon, D.
"Inside MASSIVE-3: flexible support for data consistency and world structuring"
3th International Conference on Collaborative Virtual Environments (CVE'00), ACM Press
pp.119-127, San Francisco (USA), Sep.2000.
ISBN 1-58113-303-0

[Grudin-94] Grudin, Jonathan
"CSCW: History and Focus"
IEEE Computer, IEEE Computer Society Press
v.27, n.5, pp.19-26, 1994

[Grundy-02] Grundy, J.C., Hosking, J.G.
"Engineering plug-in software components to support collaborative work"
Software - Practice and Experience, John Wiley & Sons
v.32, n.10, pp.983-1013, 2002

[Gudwin-98] Gudwin, R., Gomide, F.
"Object networks – a modeling tool"
The 1998 IEEE International Conference on Fuzzy Systems Proceedings, 1998 IEEE World Congress on Computational Intelligence, IEEE Computer Society Press
v.1, pp.77-82, 1998

[Gutwin-02] Gutwin, C., Greenberg, S.
"A Descriptive Framework of Workspace Awareness for Real-Time Groupware"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.11, n.3-4, pp.411-446, Jan. 2002
ISSN 0925-9724

[Haake-99] Haake, J.M., Wiil, U.K., Nürnberg, P.J.
"Openness in shared hypermedia workspaces: The case for collaborative open hypermedia systems"
ACM SIGWEB Newsletter, ACM press
v.8, n.3, pp.33-45, Oct. 1999

[Hill-04] Hill, J., Gutwin, C.
"The MAUI Toolkit: Groupware Widgets for Group Awareness"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.13, n.5-6, pp.539-571, Dec. 2004
ISSN 0925-9724

[Hummes-00] Hummes, J., Merialdo, B.
"Design of extensible component-based groupware"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.9, n.1, pp.53-74, 2000
ISSN 0925-9724

[Hurfin-98] Hurfin, M., Raynal, M., Tonel, F.
"A Practical Building Block for Solving Agreement Problems in Asynchronous Distributed Systems"
IEEE International Performance, Computing, and Communications Conference, IEEE Computer Society Press
pp.25-31, Phoenix (USA), Feb. 1998
ISBN 0-7803-4468-5

[Iqbal-03] Iqbal, R., James, A., Gatward, R.
"A practical solution to the integration of collaborative applications in academic environment"
5th International Workshop on Collaborative Editing Systems, at *8th European Conference on Computer Supported Cooperative Work* (ECSCW'03)
Helsinki (Finland), Sep. 2003

[Jackson-99] Jackson, L., Grossman, E.
"Integration of Synchronous and Asynchronous Collaboration Activities"
ACM Computing Surveys, ACM press
v.31, n.2, article no. 12, Jun. 1999
ISSN 0360-0300

[Jensen-92] Jensen, K.
"Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1"
EATCS Monographs on Theoretical Computer Science, Springer-Verlag
234pp., 1992
ISBN 3-5406-0943-1

[Johansen-88] Johansen, R.
"Groupware: Computer Support for Business Teams"
The Free Press
205p, 1988.
ISBN 0-02-916491-5

[Johnson-98] Johnson, R. T., Johnson, D. W.
"Cooperative learning and social interdependence theory"
Social Psychological Applications To Social Issues, (R. Tindale *et al.* eds.)
v.4, pp.9-36, 1998

[Kahler-00] Kahler, H., Mørch, A., Stiemerling, O., Wulf, V.
"Introduction to. the Special Issue on Special Issue on Tailorable Systems and Cooperative Work"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.9, n.1. pp.1-4, 2000
ISSN 0925-9724

[Kam-05] Kam, M.
"Livenotes: a system for cooperative and augmented note-taking in lectures"
ACM Conference on Human Factors in Computing Systems, ACM press
pp.531-540, Portland (USA), 2005
ISBN 1-58113-998-5

[Karsenty-94] Karsenty, A.
"Le collecticiel : de l'interaction homme-machine à la communication homme-machine-homme"
Technique et Science Informatiques
v.13, n.1, pp.105-127, 1994

[Keller-76] Keller, R.M.
" Formal verification of parallel programs"
Communications of the ACM, ACM Press
v.19, n.7, pp.371-384, Jul. 1976
ISSN 0001-0782

[Kiczales-96] Kiczales, G.
"Beyond the black box: open implementation"
IEEE Software, IEEE Computer Society Press
v.13, n.1, pp.8-11, 1996.
ISSN 0740-7459

[Kilgore-03] Kilgore, R. Chignell, M., Smith, P.
"Spatialized audioconferencing: what are the benefits?"
Conference of the Centre for Advanced Studies on Collaborative research IBM Centre for Advanced Studies Conference, IBM press.
pp.135-144, Toronto (Canada), Oct. 2003

[Kordon-05] Kordon, F., Pautet, L.
"Toward Nex-Generation Middleware?"
IEEE Distributed Systems Online, IEEE Computer Society Press
v.6, n.3, Mar. 2005
ISSN 1541-4922

[Lamport-78] Lamport, L.
"Time, Clock and the ordering of events in a distributed system"
Communication of the ACM, ACM press
v.21, n.7, pp.125-133, Jul. 1978

[Laramee-02] Laramee, T.
"Should you go with JMS?"
Java World,
Oct. 2002
http://www.javaworld.com/javaworld/jw-10-2002/jw-1025-jms_p.html

[Laurillau-02] Laurillau, Y.
"Conception et réalisation logicielles pour les collecticiels centrées sur l'activité de groupe : le modèle et la plate-forme Clover"
Thèse de doctorat Informatique
Université Joseph Fourier - Grenoble I, Grenoble,
France, Sep. 2002

[Lévy-90] Lévy, P.
Les technologies de l'intelligence, Éditions La découverte
233pp., Jan. 1990

[Lewis-04] Lewis, G., Wrage, L.
"Approaches to Constructive Interoperability"
Technical Report CMU/SEI-2004-TR-020
Software Engineering Institute, Carnegie Mellon University
59pp., Pittsburgh (USA), Dec. 2004
http://www.sei.cmu.edu/pub/documents/04.reports/pdf/04tr020.pdf

[Li-99] Li, W., Wang, W., Marsic, I.
"Collaboration transparency in the DISCIPLE framework"
ACM Conference on Supporting Group Work (GROUP'99), ACM press
pp.326-335, Phoenix (USA), 1999
ISBN 1-58113-065-1

[Lobo-99] Lobo, J., Bhatia, R., Naqvi, S.
"A policy description language"
National Conference of the American Association for Artificial Intelligence (AAAI)
pp.291-298, Orlando (USA), Jui. 1999.

[Lonchamp-03] Lonchamp, J.
Le travail coopératif et ses technologies, Hermès Lavoisier,
318pp., 2003
ISBN 2-7462-0668-4

[Lorcy-00] Lorcy, S.
"Infrastructure logicielle pour la gestion de la cohérence et de la qualité de service d'un environnement à objets réparti : application au télétravail coopératif"
Thèse de doctorat Informatique
Université de Rennes 1, Rennes
France, Jan. 2000

[Malone-95] Malone, T.W., Lai, K.Y., and Fry, C.
"Experiments with Oval: A Radically Tailorable Tool for Cooperative Work"
ACM Transactions on Information Systems, ACM press
v.13, n.2 , pp.177-205, 1995

[Mangematin-96] Mangematin, V.
"The simultaneous shaping of organization and technology within cooperative agreements"
Technological Collaboration: The Dynamics of Cooperation in Industrial Innovation (Coombs, R., Saviotti, P.P., Richards, A. eds.), Edward Elgar Publishing
pp. 119-141, Jan.1996
ISBN 1-8589-8235-9

[Merlin-76] Merlin, P.M., Farber, D.J.
"Recoverability of Communication Protocols: Implications of a theoretical Study"
IEEE Transactions on Communications, IEEE Computer Society Press
pp.1036-1043, v.24, n.9, 1976

[Microsoft-96] Microsoft Corporation
"Microsoft Corporation: DCOM Technical Overview"
White Paper, Nov. 1996
http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomtec.asp

[Mills-92] Mills, D.L.
"Network Time Protocol (Version 3) Specification, Implementation and Analysis"
Network Working Group, RFC-1305
Mar. 1992
http://www.ietf.org/rfc/rfc1305.txt

[Molina-Espinosa-04] Molina-Espinosa, J.M., Fanchon, J., Drira, K.
"Recoverability of Communication Protocols: Implications of a theoretical Study"
Third International School and Symposium in Advanced Distributed Systems (ISSADS 2004), LNCS 3061, Springer
pp.158-169, Guadalajara (Mexico), Jan. 2004
ISBN 3-540-22172-7

[Mora-02] Mora, M., Forgionne, G.A., Gupta, J.N.D.
Decision making support systems: achievements, trends, and challenges for the new decade, Idea Group Pub
418 pp., Hershey (USA), 2002
ISBN 1-59140-045-7

[Mørch-97] Mørch, A.
"Three levels of end-user tailoring: customization, integration, and extension Computers and design in context"
Computers and Design in Context, (Kyng, M. et Mathiassen, L. eds), MIT Press
pp 51-76, 1997
ISBN:0-262-11223-X

[Morgan-97] Morgan, B.
"Distributed Object Alternatives"
Java World, 1997
http://www.javaworld.com/jw-10-1997/jw-10-corbajava.html

[MSDN-02] Microsoft Developer Network
"No-Touch Deployment in the .NET Framework"
Jun. 2002
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vbtchno-touchdeploymentinnetframework.asp

[Munson-96] Munson, J., Dewan, P.
"A concurrency control framework for collaborative systems"
ACM conference on Computer supported cooperative work (CSCW'96), ACM press
pp.278-287 , Boston (USA),1996.
ISBN:0-89791-765-0

[Mwanza-01] Mwanza, D.
"Where Theory meets Practice: A Case for an Activity Theory based Methodology to guide Computer System Design "
Eighth IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT'2001), IOS Press
Jul. 2001

[OASIS-04] OASIS – Organization for the Advancement of Structured Standards
UDDI Version 3.0.2
UDDI Spec Technical Committee Draft, Dated 20041019
http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[OMG-02] OMG – Object Management Group
"CORBA Components, Version 3"
OMG document formal/02-06-65, Jun. 2002.
http://www.omg.org/cgi-bin/doc?formal/02-06-65

[OMG-04] OMG – Object Management Group
"Common Object Request Broker Architecture: Core Specification"
OMG Document formal/04-03-01, Mar. 2004.
http://www.omg.org/docs/formal/04-03-01.pdf

[Palen-03] Palen, L. Grudin, J.
"Discretionary adoption of group support software: lessons from calendar applications"
Implementing collaboration technologies in industry: case examples and lessons learned (Bjørn Erik Munkvold ed.), Springer-Verlag
pp.159-179, 2003
ISBN 1-85233-418-5

[Pekkola-00] Pekkola, S., Robinson, M., Saarinen, M.-J.O., Korhonen, J., Hujala, S., Toivonen, T.
"Collaborative Virtual Environments in the Year of the Dragon"
3th International Conference on Collaborative Virtual Environments (CVE'00), ACM Press
pp.11-18, San Francisco(USA), Sep. 2000
ISBN 1-58113-303-0

[Perrey-03] Perrey, R., Lycett, M.
"Service-oriented architecture"
Symposium on Applications and the Internet Workshops (SAINT 2003), IEEE Computer Society Press
pp.116-119, Orlando (USA), Jan. 2003
ISBN: 0-7695-1873-7

[Pierce-04] Pierce, M., Fox, G.
"Making Scientific Applications as Web Services"
Computing in Science & Engineering, IEEE Computer Society Press
v.6, n.1, pp. 93-96, 2004

[Pinelle-03] Pinelle, D., Gutwin, C., Greenberg, S.
"Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration"
ACM Transactions on Computer-Human Interaction (TOCHI), ACM press
v.10 , n.4, pp.281-311, Dec. 2003

[Prakash-99] Prakash, A
"Group Editors"
Computer-Supported Cooperative Work, Trends in Software, (Beaudouin-Lafon eds.), John Wiley & Sons
pp.103-133, 1999

[Prinz-99] Prinz, W.
"NESSIE: an awareness environment for cooperative settings"
6th European Conference on Computer Supported Cooperative Work (ECSCW'99), Kluwer Academic Publishers
pp.391-410, Copenhagen (Denmark), 1999
ISBN 0-7923-5947-X

[Ramduny-97] Ramduny, D., Dix., A
"Why, What, Where, When: Architectures for Cooperative work on the WWW"
7th International Conference on Human-Computer Interaction (HCI'97), Springer-Verlag
pp.283-301, 1997.
ISBN: 3-540-76172-1

[Richardson-98] Richardson, T.; Stafford-Fraser, Q.; Wood, K.R. et A.
"Virtual Network Computing"
IEEE Internet Computing, IEEE Computer Society Press
v.2, n.1, Jan./Feb. 1998

[Rodríguez-03] Rodríguez Peralta, L.M.
"Service de gestion de session orienté modèle pour des groupes collaboratifs synchrones"
Thèse de doctorat Informatique et Télécommunications
Institut National Polytechnique, Toulouse
France, Avril 2003.

[Roseman-93] Roseman, M., Greenberg, S.
"Building Flexible Groupware through Open Protocols"
ACM Conference on Organizational Computing Systems, ACM press
pp.279-288, Milpitas (USA), 1993
ISBN 0-89791-627-1

[Roseman-96] Roseman, M., Greenberg, S.
"Building Real-Time Groupware with GroupKit, A Groupware Toolkit"
ACM Transactions on Computer-Human Interaction, ACM press
v.3, n.1, pp.66-106, 1996.

[Roseman-97] Roseman, M., Greenberg, S.
"Simplifying component development in an integrated groupware environment"
10th annual ACM symposium on User interface software and technology, (UIST '97), ACM press
pp.65-72, Banff (Canada), 1997
ISBN 0-89791-881-9

[Rosenberg-02] J. Rosenberg et al.
"SIP: Session Initiation Protocol"
RFC 3261, Internet Engineering Task Force
June 2002
http://www.ietf.org/rfc/rfc3261.txt

[Sadani-05] Sadani, T., Saqui-Sannes, P. De, Courtiat., J.P.
"From RT-LOTOS to Time Petri Nets New Foundations for a Verification Platform"
Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05), IEEE Computer Society Press
pp.250-260, Koblenz (Germany), Sep. 2005

[Schantz-02] Schantz, R.E., Schmidt, D.C.
"Research Advances in Middleware for Distributed Systems: State of the Art"
Proceedings of the IFIP 17th World Computer Congress - TC6 Stream on Communication Systems: The State of the Art
v.220 archive pp.1-36, 2002
ISBN 1-4020-7168-X

[Schmidt-92] Schmidt, K., Bannon, L.
"Taking CSCW Seriously: Supporting Articulation Work"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.14, n.1, p7-40, 1992
ISSN 0925-9724

[Schmidt-96] Schmidt, K., Rodden, T.
"Putting it all together: Requirements for a CSCW platform"
The Design of Computer Supported Cooperative Work and Groupware Systems (Shapiro, D., Tauber, M., Traünmuller, R. eds.)
pp.157-176, Amsterdam (Netherlands),1996

[Schuckmann-96] Schuckmann, C., Kirchner, L., Schmmer, J., Haake, J. M.
"Designing object-oriented synchronous groupware with COAST"
ACM Conference on Computer Supported Cooperative Work (CSCW'96), ACM press
pp.30-38, Boston (USA), Nov. 1996

[Shen-02] Shen, H., Sun, C.
"Flexible notification for collaborative systems"
ACM Conference on Computer Supported Cooperative Work (CSCW'02), ACM press
pp.77-86, New Orleans (USA), 2002
ISBN:1-58113-560-2

[Shen-04] Shen, C. Vernier, F.D. Forlines, C., Ringel, M.
"DiamondSpin: an extensible toolkit for around-the-table interaction"
ACM Conference on Human Factors in Computing Systems (CHI'04), ACM press
pp.167-174, Vienna (Austria), 2004
ISBN 1-58113-702-8

[Slagter-00] Slagter, R.J., ter Hofte, G.H., Stiemerling, O.
"Component-based Groupware: An Introduction"
ACM Conference on Computer Support Cooperative Work (CSCW'00), *Workshop on Component-based Groupware* (CBG2000)
pp.9-15, Philadelphia (USA), Dec. 2000

[Slagter-01] Slagter, R., ter Doest, H.
"The CoCoWare Component Architecture"
TI/RS/2001/006 (GigaCSCW/D2.1.4), Telematica Instituut
104pp., Enschede (Netherlands), 2001.
https://doc.freeband.nl/dscgi/ds.py/Get/File-11636/GigaCSCWD214.pdf

[Spellman-97] Spellman, P. J., Mosier, J. N., Deus, L. M., Carlson, J. A.
"Collaborative virtual workspace"
ACM Conference on Supporting Group Work (GROUP'97), ACM Press
pp.197-203, Phoenix (USA), 1997
ISBN 0-89791-897-5

[Stal-02] Stal, M.
"Web services: Beyond component-based computing"
Communications of the ACM, ACM press
v.45, n.10, pp.71-76, 2002

[Stiemerling-00] Stiemerling, O., Cremers, A.B.,
"The EVOLVE Project: Component-Based Tailorability for CSCW Applications"
AI & Society, Springer-Verlag London
v.14, n.1, pp.120-141, 2000
ISSN 0951-5666

[Sun-03] Sun Microsystems Inc.
Java™ Remote Method Invocation Specification
2003
http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmi-title.html

[Szyperski-02] Szyperski, C., Gruntz, D., Murer, S.
Component Software – Beyond Object-Oriented Programming (Second Edition), Addison-Wesley and ACM Press
589 pp., 2002
ISBN 0-201-74572-0

[Tang-92] Tang, J.C., Isaacs, E.
"Why do users like video? Studies of multimedia-supported collaboration"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.1, n.3, pp.163-196, Sep. 1992
ISSN 0925-9724

[Tarpin-Bernard-98] Tarpin-Bernard, F., David, B.T., Primet, P.
"Frameworks and Patterns for Synchronous Groupware: AMf-C Approach"
IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction
v.150, Heraklion (Greece), Sep. 1998.
ISBN 0-412-83520-7

[Teege-96] Teege, G.
"Object-Oriented Activity Support: A Model for Integrated CSCW Systems"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
v.5, n.1, pp.93-124, 1996
ISSN 0925-9724

[terHofte-96] ter Hofte, G.H., van der Lugt, H.J., Bakker, H.
"A CORBA platform for component groupware"
OzCHI'96 Workshop on the Next Generation of CSCW Systems
pp.31-36, Hamilton (New Zealand), Nov. 1996

[terHofte-98] ter Hofte, G.H.,
"Working Apart Together : Foundations for Component Groupware"
Telematica Instituut Fundamental Research Series
No.001 (TI/FRS/001), Enschede (Netherlands), 1998
ISBN 90-75176-14-7

[Terzis-99] Terzis, S., Nixon P.
"Building the next generation groupware: A survey of groupware and its impact on the virtual enterprise"
Technical Report TCD-CS-1999-08,Department of Computer Science, Trinity College
Dublin (Ireland), 1999
https://www.cs.tcd.ie/publications/tech-reports/tr-index.99.html

[Tse-04] Tse, E., Greenberg, S.
Rapidly Prototyping Single Display Groupware through the SDGToolkit
Fifth Australasian User Interface Conference, In Conference in Research and Practice in Information Technology (CRPIT)
pp.101-110, Dunedin (New Zealand), 2004

[W3C-01] W3C – World Wide Web Consortium
"Web Services Description Language (WSDL) 1.1"
W3C Note 15 March 2001
http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[W3C-03] W3C – World Wide Web Consortium
"SOAP Version 1.2 Part 1: Messaging Framework"
W3C Recommendation 24 June 2003
http://www.w3.org/TR/soap12-part1/

[W3C-04] W3C – World Wide Web Consortium
"Requirements for the Internationalization of Web Services"
W3C Working Group Note 16 November 2004
http://www.w3.org/TR/ws-i18n-req/

[W3C-05a] W3C – World Wide Web Consortium
"XML-binary Optimized Packaging"
W3C Recommendation 25 January 2005
http://www.w3.org/TR/xop10/

[W3C-05b] W3C – World Wide Web Consortium
"SOAP Message Transmission Optimization Mechanism"
W3C Recommendation 25 January 2005
http://www.w3.org/TR/soap12-mtom/

[W3C-05c] W3C – World Wide Web Consortium
"Resource Representation SOAP Header Block"
W3C Recommendation 25 January 2005
http://www.w3.org/TR/soap12-rep/

[Wang-00] Wang, W., Haake, J.M.
"Tailoring Groupware: The Cooperative Hypermedia Approach"
Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers
Vol.9, no.1, pp. 123-146, 2000
ISSN 0925-9724

[Wilson-88] Wilson, P.
"Key Research in Computer Supported Cooperative Work, Research into Networks and Distributed Applications"
Proceedings of the European Teleinformatics Conference (EUTECO'88), (Speth, R. ed.) Elsevier Science Publishers
pp.211-226, Vienna (Austria), Apr. 1988

[Yang-04] Yang, Y., Li, D.
"Separating data and control: support for adaptable consistency protocols in collaborative systems"
ACM Conference on Computer Supported Cooperative Work (CSCW'04), ACM press
pp.11-20, 2004
ISBN:1-58113-810-5

Références Web

[.NET] http://www.msdn.microsoft.com/netframework/

[Activeworlds] http://www.activeworlds.com/

[AG] http://www.accessgrid.org/

[Apache] http://www.apache.org/

[ApacheSOAP] http://ws.apache.org/soap/

[ASTI] http://www.asti.asso.fr/

[Axis] http://ws.apache.org/axis/

[Babylon] http://visopsys.org/andy/babylon/index.html

[Blaxxun] http://developer.blaxxun.com/

[BusinessWorks] http://www.tibco.com/software/business_integration/businessworks.jsp

[COM] http://www.microsoft.com/com

[EJB] http://java.sun.com/products/ejb/

[FI] http://asn1.elibel.tm.fr/xml/finf.htm

[Foresight] http://www.foresight.org/

[Groove] http://www.groove.net/

[H.323] http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.323

[J2EE] http://java.sun.com/javaee/index.jsp

[Java] http://java.sun.com/

[JBuilder] http://www.borland.com/jbuilder/

[JMS] http://java.sun.com/products/jms/

[JOnAS] http://jonas.objectweb.org/

[JSDT] https://jsdt.dev.java.net/

[JWS] http://java.sun.com/products/javawebstart/

[KnowEdge] http://www.knowedge.net/

[Lab@Future] http://www.labfuture.net/

[LiveMeeting] http://www.livemeeting.com/

[Lotus] http://www.ibm.com/software/lotus/

[MySQL] http://www.mysql.com/

[NetBeans] http://www.netbeans.org/

[OMA] http://www.omg.org/oma/

[OMG] http://www.omg.org/

[OpenAdaptor] http://www.openadaptor.org

[OpenEAI] http://www.openeai.org

[Orbix] http://www.iona.com/products/orbix/

[OSI] http://www.opensource.org/

[Pastry] http://freepastry.rice.edu/

[PG] http://www.parallelgraphics.com/

[Platine] http://www.laas.fr/PLATINE/

[Rational] http://www.rational.com/

[RMI] http://java.sun.com/products/jdk/rmi/

[Scribe] http://freepastry.rice.edu/SCRIBE/

[SDL] http://www.sdl-forum.org/

[SharedCalen] http://www.pcdotcom.com/shared_calendar/

[Skype] http://www.skype.com/

[Telelogic] http://www.telelogic.com/

[Tomcat] http://jakarta.apache.org/tomcat/

[UML] http://www.uml.org/

[vat] http://www-nrg.ee.lbl.gov/vat/

[vic] http://www-nrg.ee.lbl.gov/vic/

[Visibroker] http://www.borland.com/us/products/visibroker/

[VRML] http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

[W3C] http://www.w3.org/

[WebLogic] http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/integrate/

[WebMentor] http://www.avilar.com/solutions/lms.htm

[WMQ] http://www-306.ibm.com/software/integration/wmq/

[XML] http://www.w3.org/XML/

[XMLSchema] http://www.w3.org/XML/Schema

LEICA : Un environnement faiblement couplé pour l'intégration d'applications collaboratives

Dans le domaine du Travail Coopératif Assisté par Ordinateur (TCAO) la recherche s'interroge toujours sur les fondements à adopter lors du développement d'applications collaboratives, aussi dites collecticiels, capables de supporter et de répondre au mieux aux besoins des utilisateurs. Outre la multidisciplinarité associée à ce domaine, le fait que les activités collaboratives impliquent plusieurs personnes exprimant des besoins différents et souvent imprévisibles de travail en groupe imposent des fortes exigences en terme d'évolutivité et flexibilité pour les applications collaboratives.

Manque d'un environnement collaboratif intégré assez ouvert, extensible et reconfigurable pour répondre à ces exigences, différentes applications collaboratives doivent être parallèlement employées pour réaliser de manière effective un travail en groupe. Bien qu'elles soient utilisées pour accomplir une tâche collaborative commune, ces applications sont exécutées de manière indépendante, sans profiter réellement les unes des autres. L'intégration de telles applications permettrait de les faire interagir de manière dynamique tout en combinant de manière contrôlée leurs fonctionnalités.

Cette thèse a donc pour objectif de concevoir un nouvel environnement pour rendre possible l'intégration de collecticiels existants tout en évitant de considérer des détails internes à ces systèmes. Cet environnement, que nous avons appelé LEICA (*Loosely-coupled Environment for Integrating Collaborative Applications*) définit une approche générale d'intégration faiblement couplée qui s'appuie sur la technologie des services Web, sur un système de notification d'événements, et sur des politiques de collaboration pour contrôler les interactions entre les applications intégrées.

Nous réalisons tout d'abord une description informelle de l'approche générale d'intégration, où les applications intégrées sont initialement contactées à travers leurs interfaces de services Web et interagissent par la suite en échangeant des notifications d'événements. Leurs interactions sont contrôlées par la politique de collaboration spécifiée pour une session de travail définissant comment l'activité collaborative supportée par une application est affectée par l'information reçue d'une ou plusieurs autres applications. Nous spécifions de façon détaillée l'architecture de LEICA permettant de mettre en œuvre une telle approche d'intégration. Nous proposons ensuite une méthode pour formaliser et valider cette architecture au moyen du profil UML/SDL et du module de simulation supportés par l'outil *TAU G2* de *Telelogic*. Un premier prototype de LEICA a été également implémenté et deux applications collaboratives se trouvent actuellement intégrée à l'environnement.

LEICA: Loosely-coupled Environment for Integrating Collaborative Applications

In the Computer Supported Cooperative Work (CSCW) domain, researchers have always wondered about which concepts and architectures to adopt for the development of collaborative applications (or groupware), capable to suitably meet user requirements. Besides the multidisciplinarity inherent to this domain, considering that collaborative activities engage several people presenting different (often unpredictable) needs, evolutivity and flexibility appear as two mandatory requirements for collaborative applications.

However, integrated collaborative environments are rarely open, extensible and reconfigurable enough so as to meet these requirements. As a result, users decide to "create" their own collaborative environments by using different collaborative applications, working side by side but independently, without really getting advantage of each other. Allowing the integration of these applications could bring significant benefits to users. An integrated collaboration environment would allow different functionalities of existing applications to be dynamically combined and controlled (enhancing therefore flexibility).

This thesis' main goal is to design a new environment allowing the integration of existing groupware, without dealing with their low-level features. This environment, called LEICA (Loosely-coupled Environment for Integrating Collaborative Applications) defines a loosely-coupled integration approach which is based on Web services technology, an event notification system, and the definition of collaboration policies to control the interactions among integrated applications.

We start from an informal description of the general integration approach, where integrated applications are initially contacted through their Web services interfaces, interacting thereafter by exchanging event notifications in the context of an integrated collaborative session. Their interactions are controlled by the collaboration policy defined for this collaborative session. The collaboration policy defines how the collaboration activity supported by one application will be affected by information received from other applications. After that, we describe in this thesis the architecture of LEICA enabling the implementation of such an integration approach. Based on this informal architectural description, we propose a method for formalizing it by using the UML/SDL profile and the simulation functionalities supported by the *Telelogic* software tool Tau G2. Finally, we have also implemented the first prototype of LEICA where two collaborative applications are currently integrated.