Service d'agrégation d'événements dans une plateforme coopérative

Sawsan Alshattnawi

Université – Nancy 2

Service d'agrégation d'événements dans une plateforme coopérative

MÉMOIRE

22 juin 2005

pour l'obtention du

DEA Informatique de Lorraine -École Doctorale IAEM Lorraine

par

Sawsan ALSHATTNAWI

Composition du jury

	Dominique Méry	Professeur UHP-NancyI et ESIAL
	Didier Galmiche	Professeur UHP-NancyI
	Noëlle Carbonell	Professeur UHP-NancyI
	Olivier Festor	Directeur de recherche INRIA
Encadrant :	Gérôme CANALS	Maître de Conférences à l'Université Nancy2

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

à mes parents,
à Mohammed,
à Naseraldeen et Raheeg

Remerciement

Je remercie ceux qui m'ont apporté leur soutien pendant ce stage de DEA en particulier :

- *Les membres du jury*

  Pour avoir accepté de juger pour avoir accepté de juger mon travail et pour leurs presences jusqu'à ce jour.

- *Mon encadrant*

  Pour ses nombreux conseils.

- *Mon chef*

- *Florent Jouille*

  Pour son aide concernant le projet libresource.

- *Tous Les membres de l'équipe ECOO et Mes collègues*

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Contexte du stage

L'un des services de base de toute plateforme coopérative est le *service de conscience de groupe* qui permet aux participants de se tenir au courant des activités du groupe. Ce type de service offre notamment des mécanismes pour suivre l'activité des autres participants ainsi que l'état de l'espace de travail partagé. Il offre la possibilité de percevoir et comprendre l'action des autres et ainsi de mieux coordonner ses propres initiatives avec l'ensemble du groupe. Il offre également la possibilité de percevoir et comprendre l'avancement global du travail et ainsi de situer ses propres initiatives par apport à l'objectif global.

Ce type de service se base sur l'utilisation et la diffusion d'événements produits par la plateforme supportant l'activité coopérative, et leurs mise à disposition des utilisateurs sous une forme compréhensible. Des nombreux travaux ont déjà été réalisés sur ce sujet depuis une quinzaine d'années.

Quand on travaille ensemble sur un projet commun en utilisant une plateforme de collaboration, les événements qui se déclenchent sur cette plateforme sont nombreux. Le nombre d'événements augmente avec la complexité du projet : taille du groupe de participants, nombre d'objets et de documents manipulés, nombre d'activités mises en jeu, grande variété des rôles au sein de l'organisation.

Cette complexité conduit en particulier à :
- Submerger les participants de notifications et d'informations, rendant ainsi très complexe la compréhension globale de l'activité du groupe par chaque participant,
- diminuer la pertinence des informations offertes aux utilisateurs, qui reçoivent des informations qui ne les concernent pas directement au moment où ils les reçoivent.

Une part des travaux effectués jusqu'à présent sur le travail coopératif ont visés à résoudre ce problème en construisant des outils qui diminuent les nombres de notification message et présentent aux participants des informations plus adaptés à leur travail. En particulier, une approche a été proposée dans la thèse de Christophe Bouthier [Bouthier, 2003]. Cette approche se base sur la reconnaissance et la prise en compte du contexte d'activité des différents participants d'un projet pour :

- agréger les événements à la source, pour obtenir des informations moins nombreuses mais

de plus haut niveau et plus compréhensibles,
- diffuser ces informations de manière plus selective,
- adapter la présentation des informations à la situation individuelles des participants, en faisant varier notamment le niveau de détail et le moment de la présentation qui peut être retardé.

Cette approche est basée sur une architecture d'égal-à-égal. L'outil de conscience de groupe proposé est réparti sur l'ensembles des machines des participants d'un projet. Chaque site est chargé de détecter, agréger et diffuser les informations locales, et recevoir, adapter et présenter les informations en provenance des autres participants.

1.2 Sujet

Notre objectif est de proposer une approche de conscience de groupe traitant le problème de la surcharge d'informations dans le cas d'une plateforme de coopération basée sur un serveur et des clients répartis. A la différence d'une architecture purement pair-à-pair, une plateforme collaborative client/serveur centralise un certains nombre d'informations sur l'état courant du projet : structure organisationnelle, activités et tâches, documents partagés. Ces informations sont particulièrement utiles pour la conscience de groupe et sa mise en oeuvre.
Notre approche est basé sur l'adaptation du modèle proposé dans la thèse de C. Bouthier au cas d'une plateforme collaborative Client/Serveur.
Notre travail est concentré sur l'étape d'agrégation pour agréger les événements qui se déclenchent sur le serveur. La plateforme d'expérimentation est la plateforme LibreSource développée dans l'équipe ECOO. Les développements consistent à écrire une ressource libresoource en Java.

1.3 Présentation du document

La mémoire comprend trois chapitres :
Le chapitre deux est consacré à la problématique abordée dans ce travail. Il présente notamment des définitions sur la conscience de groupe et la conscience de groupe en contexte. La plateforme LibreSource et son architecture sont présentées et les problèmes à résoudre pour atteindre notre objectif sont explicités.

Le chapitre 2 est consacré à la proposition d'une nouvelle architecture fonctionnelle. Cette architecture fonctionnelle reprend les propositions de la thèse de C. Bouthier en les adaptant au cas d'une plateforme Client/Serveur. Le chapitre donne d'abord des détails sur l'architecture de départ et présente les propositions d'extensions nécessaire à la prise en compte d'un serveur centralisé.

Le chapitre 3 est dédié à la conception d'un module pour l'Agrégation des événements dans la plateforme coopérative centralisé. L'approche choisie pour réaliser cette agrégation est différente de celle utilisée dans l'approche initiale est se base sur comment l'agrégation s'est faite dans le réseau égal à égal et comment nous avons utilisé une autre technologie plus pertinente à la plateforme coopérative qui s'appelle le traitement d'événements complexes "complex event processing" (CEP), comment le langage RAPIDE est utilisé pour le CEP et puis on adapte cette approche aux événements dans le LibreSource. Ensuite sera présenté la stratégie d'implementation et notre action sur la programmation et ses résultats.

Chapitre 2

Problématique

2.1 La Collaboration

2.1.1 Définition

La collaboration est le travail interactif entre au moins deux personnes dans le but de communiquer, partager des ressources, coordonner, résoudre des problèmes, réaliser un projet ou participer à une négociation [sap, 2005]. La collaboration peut se dérouler dans une réunion face-à-face mais à présent, on essaie de coordonner le travail lorsque les personnes sont réparties dans le temps et dans l'espace, en utilisant une infrastructure logicielle. C'est la responsabilité du CSCW (Computer Supported Cooperative Work), ou TCAO (Travail Coopératif Assisté par l'Ordinateur). Le CSCW est une discipline de l'informatique qui a pour objet la compréhension, l'organisation et le support du travail coopératif médiatisé par ordinateur. Elle étudie les méthodologies du travail collaboratif entre des personnes travaillant ensemble, comment les technologies informatiques et associées trouvent leur place dans ce cadre[Fitzpatrick et al., 1999].

2.1.2 Exemples

Le logiciel BSCW, "Basic Support for Cooperative Work projet" [Bentley et al., 1995] est un exemple classique de système coopératif. BSCW est un système extensible pour partager et retrouver des informations.
Il consiste en un serveur, accessible de différentes plateformes, qui gère plusieurs espaces de travil (workspaces). Chaque workspace contient des objets partagés comme des documents, des URLs, des répertoires, ainsi que des membres et des groupes qui peuvent accéder à ces objets. Le serveur BSCW est implanté au dessus d'un serveur Web http, il gère l'ensemble des espaces de travail et contrôle les accès à ces espaces de travail. Pour travailler, un utilisateur doit s'identifier auprès du serveur qui lui propose alors de choisir l'espce de travail qui lui convient (au cas où il a des droits sur plusieurs espaces de travail).

BSCW propose un service de conscience de groupe basé sur les événements et la notification. Un événement est produit à chaque changement d'état d'un espace de travail. L'ensemble des membres connectés à cet espace de travail reçoit alors une notification signalant ce changement d'état.

2.1.3 La conscience de groupe

Le terme *conscience de groupe* (group awareness) désigne l'ensemble des informations et mécanismes permettant à des personnes de percevoir et comprendre toutes les évolutions de l'espace partagé de collaboration. La conscience de groupe recouvre des informations sur : où les autre travaillent, ce qu'ils font, et ce qu'ils vont faire[Gutwin and Greenberg, 2002]. Ces informations permettent aux membres du groupe de participer effectivement aux activités collaboratives, et de coordonner leurs actions conjointes. Dans le cas où ces personnes sont dans un même lieu et au même moment (collaboration directe non médiatisée), ces informations sont aisément obtenues. Dans la situation où les personnes sont réparties dans l'espace ou dans le temps, la collection et la présentation de ces informations est plus difficile et nécessitent un mécanisme logiciel spécifique et intégré à la plateforme de collaboration.
Beaucoup de modèles et de systèmes ont émergé depuis une quinzaine d'années pour proposer des approches qui collectent et présentent les informations de la conscience de groupe de manière pertinente. On peut citer tout d'abord les travaux autour des *mediaspaces*, dont le système PortHoles [Dourish and Belloti, 1992] est le premier représentant. Ce type de système base la conscience de groupe sur l'intégration d'images vidéos sur l'écran informatique. Ces images permettent de suivre l'activité des autres membres du groupe.
Une autre classe de systèmes propose des approches à base de *widgets graphiques* spécialisés pour l'interaction en situation coopérative [Gutwin and Greenberg, 1996],[Gutwin et al., 1996]. Cette approche propose d'intégrer dans l'interface d'accès à l'espace de travail des éléments pour visualiser l'action des membres du groupe.
Une troisième voie correspond à des infrastructures pour la détection et la diffusion d'événements, comme Elvin [Fitzpatrick et al., 1999] ou Nessie[Prinz, 1999]. Chaque événement produit correspond à un changement d'état dans l'espace collaboratif. Il faut noter aussi des travaux plus théoriques visant à proposer un modèle de la conscience de groupe. Dans [Rodden, 1996], il est proposé un modèle basé sur la métaphore spatiale : les éléments constitutifs de la conscience de groupe (objets, participants) sont placés dans un espace dans lequel on peut calculer une distance permettant de déterminer ce qui est vu par les participants de la coopération. Ce type de modèle permet de modéliser la pertinence des informations pour les membres de la coopération en se basant sur la notion de distance.

Actuellement, des efforts sont faits pour améliorer la pertinence des informations offertes aux utilisateurs par la prise en compte de leur contexte d'activité. Par exemple, le système ENI (Event and Notification Infrastructure) [Gross and Prinz, 2003] qui combine des événements logiques de l'espace logiciel et des événements capturés dans l'espace physique, pour reconnaître et prendre en compte la situation d'usage dans la présentation des informations aux utilisateurs.

Les éléments de la conscience de groupe

La conscience de groupe consiste en un ensemble d'éléments que les participants doivent connaître [Gutwin and Greenberg, 2002], [Fitzpatrick et al., 1999]. Ces éléments sont :

1. Qui(who) : Est-ce qu'il y a d'autres personnes dans l'espace de travail ? et qui sont-elles ?
2. Quoi (what) : Ce que font ces personnes en général ou en détail, et sur quel(s) objet(s) travaillent-elles ?
3. Où (where) : Où travaillent ces personnes ?
4. Quand (when) : Quand les événements se produisent-ils dans le système ?

5. Comment (how) : Comment l'activité se produit ?

La réponse à chaque question (*qui travaille avec nous, ce qu'ils font, où ils sont, quand et comment les événements se produisent*) nous donne des indications sur les informations que l'outil de conscience de groupe doit collecter.

Par ailleurs, [Tam and Greenberg, 2004] propose plusieurs perspectives pour analyser et structurer les informations de conscience de groupe, suivant le type de questions que l'utilisateur peut poser au système. Par exemple si le participant est intéressé par un objet, la *perspective objet* de la conscience de groupe lui permet de trouver des réponses à toutes les questions relatives à l'état et l'évolution des objets partagés. Si au contraire il est intéressé par une personne, la *perspective orienté personnes* lui offre des réponses sur des questions liés aux membres du groupe (que fait-elle ?, qu'est-ce qu'elle a fait ?, pourquoi ? et comment ?).

Les dimensions de la conscience de groupe

Certains auteurs identifient plusieurs dimensions dans la conscience de groupe. Prinz, dans [Prinz, 1999] en identifie deux principales : La conscience sociale et la conscience de l'activité.

1. La conscience sociale est l'information que l'utilisateur obtient sur ses collègues dans le contexte d'une conversation sociale. La notification pour la conscience sociale contient des informations sur : qui est là et son état dans l'environnement partagé.
2. La conscience de l'activité correspond aux informations relatives aux activités des autres membres du groupe. Par exemple, être informé de ce que les autres membres font à un moment donné. Elle permet aux utilisateurs de coordonner leurs activités sur un objet partagé. La notification contient des informations concernant qui fait l'action et sur quel objet. Interlocus [Nomura et al., 1998] est un exemple de système qui notifie à l'utilisateur tous changements sur les objets partagés.

Dans [Steinfield et al., 1999] on trouve encore les dimensions suivantes :

1. La conscience de présence : Il y a des applications qui contrôlent la présence et la disponibilité d'un membre pour faciliter l'interaction sociale. ICQ est un système par lequel l'utilisateur peut indiquer sa présence physique et son envie d'accepter l'interaction avec les autres.
2. La conscience de la perspective : les participants ne veulent pas savoir les actions des autres dans le passé mais ils sont intéressés par le moyen émergence de ces actions.
3. La conscience de l'environnement : c'est d'être au courant des événements qui se produisent dans l'espace physique. Elle est identique à la conscience sociale dans Prinz (1999).

Les problèmes de la conscience de groupe

Les approches pour la conscience de groupe proposées jusqu'à présent se sont montrées efficaces dans des contextes précis. Elles fonctionnent bien dans des groupes de petite taille, partageant peu de documents et souvent de façon synchrone. C'est par exemple le cas typique de l'édition collaborative d'un document commun.
Par contre, ces approches sont prises en défaut lorsqu'il s'agit de concevoir un système de conscience de groupe adapté au support de projet collaboratifs de grande taille, par exemple dans le cas de l'ingénierie coopérative. Ces projets se caractérisent, entre autre, par :

- une structure organisationnelle qui peut être complexe : rôles nombreux, groupes et sous-groupes nombreux, relations entre ces groupes variées,

- un nombre d'objets, de documents et de tâches qui peut être grand,
- un nombre de participants qui peut être grand, avec des situations individuelles (rôles, tâches, objets manipulés, relations sociales) très diverses.

Dans un tel cadre, les approches proposées sont mal adaptées du fait du trop grand nombre d'informations ou d'événements produits, et de la diversité des situations individuelles. Elles conduisent notamment à :

- Surcharger l'utilisateur d'informations, ce qui risque de le conduire soit à passer trop de temps pour percevoir les travail des autres, au détriment de son propre travail, soit à volontairement ne pas traiter certaines information et ainsi obtenir une vision erronée de l'état de la collaboration,
- Proposer à l'utilisateur des informations inadaptées à sa situation individuelle, qui introduisent ainsi un bruit supplémentaire dans les informations déjà nombreuses qu'il reçoit.

Pourtant, les mécanismes de conscience de groupe restent indispensables dans les projets coopératifs de grande taille.

2.1.4 La conscience de groupe contextuelle

La conscience de groupe contextuelle est une première tentative de réponse au problème de la surcharge d'information dans les plateformes pour projets coopératifs de grande taille. L'idée est d'utiliser une représentation du contexte de travail de chaque utilisateur pour adapter les informations qui lui sont offertes. L'adaptation concerne les informations et leur contenu, ainsi que leur présentation. On espère ainsi améliorer la pertinence des informations transmises à chaque utilisateur et en diminuer le volume, limitant ainsi la surcharge potentielle.
Quelques propositions ont déjà été faites dans cette voie. Par exemple dans [Gross and Prinz, 2003], le contexte de travail recouvre l'ensemble des informations caractérisant l'environnement physique d'interaction d'un utilisateur. Le système tient compte de ce contexte pour proposer un représentation des informations adaptée à la situation donnée (en réunion, en déplacement, au bureau ...).
Dans [Bouthier, 2003], le contexte de travail est limité au contexte d'activité à l'intérieur de la plateforme de collaboration (à l'exclusion du contexte physique). Par contre, ce contexte est utilisé également à la source pour agréger les événements collectés avant de les diffuser aux autres participants. Les informations transmises sont ainsi moins nombreuses et de plus haut niveau. Le contexte est ensuite utilisé pour adapter la représentation de ces informations à la situation individuelle de chaque participant. Ici, le contexte se définit comme l'ensemble des informations haut niveau reçues par l'outil de conscience de groupe. Ces informations haut niveau proviennent à la fois de l'agrégation d'événements locaux issus de l'activité de l'utilisateur, et des outils des autres membres du groupe.

Cette approche a donc un double effet sur la surcharge d'information : l'agrégation permet de diminuer le nombre d'informations échangés entre les membres du groupe, l'adaptation permet d'améliorer la pertinence des informations finalement proposées à chaque utilisateur.

Les dimensions du contexte de travail

On peut distinguer plusieurs dimensions du contexte de travail :

- le contexte géographique et la localisation (dans quel bâtiment , quel étage et quel bureau).
- le contexte organizationnel (dans quel projet et dans quel département les personnes se trouvent).

- le contexte personnel et social (dans quelle famille et quels sont ses amis) [Abowd et al., 1999].
- le temps (quand la personne se connecte et quand elle se déconnecte).
- l'activité (quelle sont les activités qu'elle a fait, son activité courante) [Gross and Prinz, 2003].
- l'outil utilisé pour interagir avec l'espace de collaboration (ordinateur de bureau, PDA, téléphone ...)

Ces différentes dimensions ont toutes une influence sur l'adaptation des informations de la conscience de groupe qu'il faut réaliser. Ainsi, de manière idéale, un outil de conscience de groupe devrait être capable de reconnaître et de prendre en compte toutes ces dimensions et facettes du contexte de travail des utilisateurs.

2.2 La plateforme coopérative LibreSource

LibreSource [Lib, 2005] est un environnement coopératif distribué offrant des services pour héberger des équipes distribuées. Il permet notammet à une équipe distribuée de s'organiser, de travailler ensemble et de coopérer à la réalisation d'un projet au travers d'un ou plusieurs espaces de travail partagés.
LibreSource combine d'une part des outils de production coopérative : service de partage de fichiers et de coordination et d'autre part des outils de gestion de communautés : services de communication et de gestion de contenu.
LibreSource est une plateforme orientée Web : les services sont accessibles au travers du protocole http via un serveur Web. Les clients permettant l'accès aux services sont donc des navigateurs web.
LibreSource est le produit d'un projet RNTL associant le projet ECOO du Loria, la société Artenum et l'Université de Paris 7.

2.2.1 Les composants de LibreSource

LibreSource est un ensemble de composants modulaires que l'on assemble au moment de la configuration d'un projet en fonction des besoin de ce projet. De nombreux composants sont classiques : forum, wiki, traceur de bugs, gestionnaire de groupes. Deux composants ont une importance particulière et sont originaux : So6 un gestionnaire de configuration d'un type nouveau, et Bonita, un système de gestion de workflow.
So6 est un outil de gestion de configurations basé sur un outil de fusion de données robuste et unifié (un algorithme commun à tous les types de données). Contrairement aux outils classiques, il n'est pas basé sur la notion de version, mais se base sur la gestion des histoires d'opérations appliquées aux données et utilise des mécanismes de transformations d'opérations pour réaliser les fusions. Bonita est un outil de gestion de flots de tâches (workflow) flexible et dynamique permettant de décrire des enchaînements d'activités et de contrôler leur execution.

Un environnement dans LibreSource est un ensemble de ressources organisé de manière arborescente. Travailler dans un tel environnement consiste à créer, éditer, modifier, déplacer, supprimer des ressources. Une ressource peut correspondre à un service instancié dans un projet : un forum, un wiki, un espace de documents, ou à une ressource de type donnée manipulée par le projet : un membre, un document, un groupe ...Le nommage des ressources est basé sur leur position dans l'arbre des ressources. Le nom et la position de la ressource dans l'arbre est déterminé au moment de sa création : la ressource créée est liée à une URI composée notamment du chemin d'accès à la ressource depuis la racine de l'arbre.

Un exemple d'arbre de ressources dans LibreSource :

```
°LibreSource
    °project
        °forum
            °mailingList
    °file
    °timeline
        °project
            °file
```

Les ressources sont classées en quatre catégories :

1. Le partage de données : transfert de fichiers, fusion de données (So6),
2. La communication : LibreSource fournit des forums, liste de diffusion et wiki page qui permet aux utilisateur de créer et éditer des page web en utilisant le navigateur web.
3. La coordination : LibreSource fournit le "bug tracker" et le gestionnaire de tâches Bonita.
4. La conscience de groupe : LibreSource gère des queues d'événements persistants produits par les ressources.

2.2.2 Les événements dans LibreSource

Un événement est un objet persistant crée dans le système chaque fois qu'une opération sur une ressource se produit. Le type de l'événement dépend de la ressource et de l'opération déclenché sur cette ressource. Ainsi, un événement permet d'identifier exactement l'opération qui s'est déroulée sur le serveur. On distingue deux types d'événements dans LibreSource :

- Général : ce sont des événements correspondant à des opérations existant pour toutes les ressources : créer, éditer, et supprimer.
- Dépendant : ce sont des événements correspondant à des opérations spécifiques à une resource.

Par exemple, les opérations, et donc les événnements, associées à la ressource "Files" sont :

```
- LibreSourceFiles.file.create
- LibreSourceFiles.file.edit
- LibreSourceFiles.file.delete
- LibreSourceFiles.file.download
```

Annex (A) contient toutes les ressources avec leurs événements dans LibreSource.

Les aspects d'événement sont trois :

- form : l'événement est un objet qui a des attributs, comme : quand il s'est produit, où il s'est produit, qui l'a fait et quelle est sa signification.
- signification : l'événement signifie une activités.
- relatif : une activité est reliée avec une autre activité soit par temps ou par causality ou par l'agrégation.

LibreSource dirige des événements, créer, déplacer, ... les clients peuvent souscrire à l'événement pour recevoir les messages de notification par email lorsqu'un événement se produit.

2.2.3 La conscience de groupe dans LibreSource

La conscience de groupe dans LibreSource est basée sur le mécanisme des événements. Les événements produits par le système sont stockées dans une queue persistante.
Les clients LibreSource peuvent souscrire à une queue d'événements et reçoivent des messages de notification pour chaque événement ajouté dans cette queue. Ce message contient les informations concernant la forme de l'événement : type, date de création, ressource concernée, origine ...

Le nombre de ces événements peut être très grand, d'une part parce que le nombre d'opérations exécutées sur le serveur peut être important, et d'autre car l'exécution d'une activité par un client peut mettre en jeu plusieurs ressources et donc générer plusieurs événements. Par exemple, l'activité de création d'un projet va correspondre aux opérations :

```
- LibreSourceCore.project.create
- kernel.create
- kernel.bind
```

2.3 Objectifs et problèmes abordés

Comme dans de nombreuses plateforme coopérative, le problème de la surcharge d'informations dans la conscience de groupe se pose dans LibreSource. L'objectif de ce travail est de proposer une solution à ce problème dans LibreSource en adaptant l'approche de la conscience de groupe contextuelle proposée dans la thèse de C. Bouthier [Bouthier, 2003].

A terme, l'objectif est donc la construction d'un mécanisme de conscience de groupe utilisant un représentation du contexte de travail des utilisateurs pour agréger les informations et les adapter aux situations individuelles. De nombreux problèmes sont à résoudre pour atteindre cet objectif, liés entre autre à la représentation et la reconnaissance du contexte de travail et aux mécanismes d'adaptation. Dans ce mémoire, nous abordons deux problèmes particuliers concourant à la réalisation de l'objectif général :

1. Adapter l'architecture : l'approche proposée dans [Bouthier, 2003] est basée sur une architecture pair-à-pair. Adapter cette approche au cas de LibreSource suppose de revisiter cette architecture pour intégrer un serveur centralisé. Ce serveur détient un grand nombre d'informations sur l'état du projet et les activités en cours, et produit des événements permettant de suivre l'activité. Le serveur a donc un rôle particulier, et ne peut pas être considéré comme les autres clients. Il faut donc proposer une architecture fonctionnelle reconnaissant ce rôle particulier.
2. Agréger les événements sur le serveur : comme nous l'avons présenté ci-dessus, le serveur produit un grand nombre d'événements. L'approche proposée dans [Bouthier, 2003] se base entre autre sur l'agrégation des événements à la source (sur les différents clients) pour en diminuer le nombre et améliorer leur niveau sémantique. Le serveur étant lui même une source importante d'événements, il est nécessaire de le doter d'un mécanisme d'agrégation de ces événements.

Les deux chapitre suivant sont consacrés à la proposition de solutions pour ces ces deux problèmes.

Chapitre 3

L'architecture fonctionnelle

3.1 Introduction

L'architecture fonctionnelle qui a été proposée dans [Bouthier, 2003] est basée sur une approche égal-à-égal : il n'y a pas de serveur centralisé jouant un rôle particulier. Les applications des utilisateurs communiquent directement et il n'y a pas de gestion centralisée des connaissances.
Au contraire, LibreSource est une plateforme basée sur un serveur centralisée détenant une grande quantité de connaissance sur l'état des projets hébergés et produisant des événements indiquant tous les changements et transitions dans l'état de ces projets.
Ce chapitre a pour objet de proposer une architecture fonctionnelle de conscience de groupe pour les plateformes coopératives utilisant une gestion centralisée de l'état des projets, à l'image de LibreSource. Cette architecture adapte à ce cadre l'approche de [Bouthier, 2003]. L'aport principal consiste à répartir les différentes étapes proposées entre le serveur et les clients, et à identifier les coordinations nécessaire entre ces étapes.

3.1.1 Groupe communication dans le Web

Puisque notre architecture adaptée à la plateforme coopérative sur Internet, les clients web peuvent coordonner leur travail et communiquer facilement. Dans le web la communication tombe dans deux types :

1. Synchronous : ce type de communication se réalise quand les utilisateurs du même groupe sont en ligne au même temps, pour que les membres sachent qui est en ligne, un message de notification est envoyé à tout le groupe chaque fois qu'un membre se connecte ou se déconnecte de sa session. Quand le membre est en ligne il peut envoyer un-ligne message à un autre membre qui aussi est en ligne[Fitzpatrick et al., 1999]. Il peut aussi faire des conversations temporel (chatting) [Steinfield et al., 1999] par un Java applet à chaque page.
2. Asynchronous : ce type passe aux utilisateurs les informations sur les changements dans le workspace par email, ou par attacher les changements récents à chaque objet dans le web page, et les membres peuvent connecter en utilisant un mailing list [Lib, 2005], un email est envoyé au liste d'adresse.

Les membres de groupe peuvent utiliser des moyens de communication pour être consciente au workspace coopératif.

3.2 Architecture Pair-à-Pair pour la conscience de groupe contextuelle

Les outils de conscience de groupe traitant des événements sont en général organisés en deux grandes parties : la partie émettrice et la partie réceptrice. La partie émettrice d'un outil collecte les informations et les diffuse aux membres du groupe. Ces informations sont reçues par les parties réceptrices des autres outils, qui présentent ces informations à leurs utilisateurs.

Dans [Bouthier, 2003], cette architecture de base est étendue par des modules utilisant une représentation du contexte de travail de l'utilisateur pour agréger et adapter les informations de conscience de groupe. Trois étapes sont ajoutées : une étape d'agrégation et une étape de filtrage sur la partie émettrice, une étape d'adaptation sur la partie réceptrice.
Ces étapes utilisent une représentation du contexte de travail de l'utilisateur fondée sur une modélisation probabiliste des tâches et du comportement de l'utilisateur. Cette modélisation permet de déduire l'activité courante d'un utilisateur à partir de l'observation des événements produits dans son environnement. Elle permet également de déduire d'autre informations sur son comportement, par exemple son degré d'attention, les relations qu'il crée avec les autres membres du groupe etc... Les informations de haut niveau déduites à partir des événements de bas niveau constituent le contexte de travail de l'utilisateur.

Cette architecture, présentée dans la figure (3.1), propose les étapes suivantes :

1. Collection : collecter des informations bas niveau sur le contexte de travail de chaque utilisateur, ces informations sont obtenues soit automatiquement par l'outil soit par interrogation de l'utilisateur. L'outil doit capturer le plus d'informations possible pouvant donner des indications sur l'activité de l'utilisateur. La collection des informations dépend aussi de la fréquence avec laquelle l'information est collectées. Certaines informations sont susceptibles de changer à tout moment, et doivent être capturées fréquemment. Par exemple la fréquence des clics souris. Certaines informations ne sont disponibles qu'avec l'ajout d'un material spécifique. C'est par exemple des sensor ou des webcam pour capturer les regards ou les gestes.
2. Agrégation : agréger les informations de l'étape précédente de telle façon à obtenir des informations réduites et de plus haut niveau.
   L'outil les agrège en répondant aux questions suivantes : où dans une seule information haut niveau.
   L'outil utilise des réseaux bayésiens [Charniak, 1991], [Horvitz et al., 1998] pour modéliser le comportement de l'utilisateur. Un réseau bayesien permet de représenter des connaissances incertaines sur un phénomène complexe. Il consiste à relier des événements observables à une ou plusieurs causes probables de ces événements. Ensuite, un outil de raisonnement probabiliste permet, à partir d'une séquence observée d'événement, de déduire les cause probables de cette séquence. Ce raisonnement revient donc à agréger un ensemble d'événements observés en leur cause probable.
   L'outil utilise plusieurs canaux d'agrégation correspondant à différentes facettes du contexte de travail : qui, pourquoi, quand, quoi, comment. Chaque événement observé est fournir à l'entré de chacun des canaux. La sortie de chaque canal produit des événements de haut niveau caractéristique chacun d'une facette du contexte.
3. Filtrage : le but de cette étape est de permettre à l'outil de choisir les informations à diffuser aux autres membres de groupe, et de diffuser des informations plus pertinentes à

chacun. La technique qui est utilisée dans [Bouthier, 2003] pour faire le filtrage est consiste à étiqueter les informations produites par l'étape d'agrégation, puis à prendre une décision. L'étiquetage est réalisé en se basant sur les informations contenues dans le contexte de travail, et pour chaque destinataire potentiel. Sur la base de ces informations, une distance est calculée. Cette distance permet de déterminer le niveau de détail des informations à transmettre au destinataire. Ce niveau peut varier de *aucun détail* à *tous les détails*.

4. Diffusion : le but de cette étape est diffuser les informations de l'étape précédente aux personnes concernées.
5. Réception : le but de cette étape est de recevoir des informations et les sauvegarder pour les passer à l'étape suivante. Lorsque l'utilisateur est déconnecté, il ne peut pas recevoir les informations haut niveau des autres membres du groupe. Ces informations sont stockées du côté de l'émetteur dans un tampon d'attente. dès que l'utilisateur est de nouveau accessible sur le réseau, toutes les informations en attente sont envoyées dans un message unique.
6. Adaptation : le but de cette étape est d'adapter les informations de l'étape précédente au contexte de travail du récepteur. Comme dans l'étape de filtrage, la technique consiste à étiqueter les informations reçues pour ensuite prendre une décision concernant la visualisation. Cet étiquetage se base sur :
   - Les informations reçues des utilisateurs distants,
   - Le contexte de travail de l'utilisateur, ces informations sont reçues de l'étape d'agrégation sur la machine locale.
   - Des informations sur la relation entre l'utilisateur récepteur et l'émetteur.

   L'étiquetage consiste à déterminer une valeur caractérisant le niveau d'intrusion à utiliser pour présenter l'information. Cinq valeurs sont prévues :
   - (-1) signifie que les informations ne sont pas pertinentes, et sont donc oubliées,
   - (0) signifie que les informations sont pertinentes mais ce n'est pas le moment pour les presenter au client ; elles sont donc conservées et la valeur est recalculée à un moment ultérieur,
   - (1,2,3) signifient que les informations sont pertinentes et doivent être présentées à l'utilisateur. La valeur désigne un degré d'intrusion à utiliser pour présenter l'information : peu intrusif (à sa demande uniquement) s'il s'agit d'une information non cruciale, de façon périphérique (par exemple dans un panneau de contrôle), présentation normale s'il s'agit d'une information pas suffisamment importante pour interrompre l'utilisateur dans sa tâche courante , ou très intrusif (par exemple une fenêtre d'alerte) s'il est justifié d'interrompre l'utilisateur dans sa tâche.
7. Présentation : en cette étape l'outil décide la façon dont l'information va être représentées en fonction du contexte de travail de l'utilisateur. Elle consiste à mettre en correspondance les valeurs d'adaptation avec un mode d'interaction avec l'utilisateur.
8. Visualisation : le but de cette étape de presenter les informations aux utilisateurs.

L'étape d'agrégation réduit le nombre de notifications en agrégeant les événements et l'étape de filtrage réduit le nombre des informations en filtrant les informations non pertinentes.

L'outil utilise le contexte de travail pour adapter la conscience de groupe à l'activité de l'utilisateur. Ce contexte de travail est utilisé à plusieurs étapes :

- L'étape d'agrégation : transformer les informations de bas niveau en informations de haut niveau moins nombreuses, caractéristiques de l'activité et du contexte de travail de l'utilisateur.

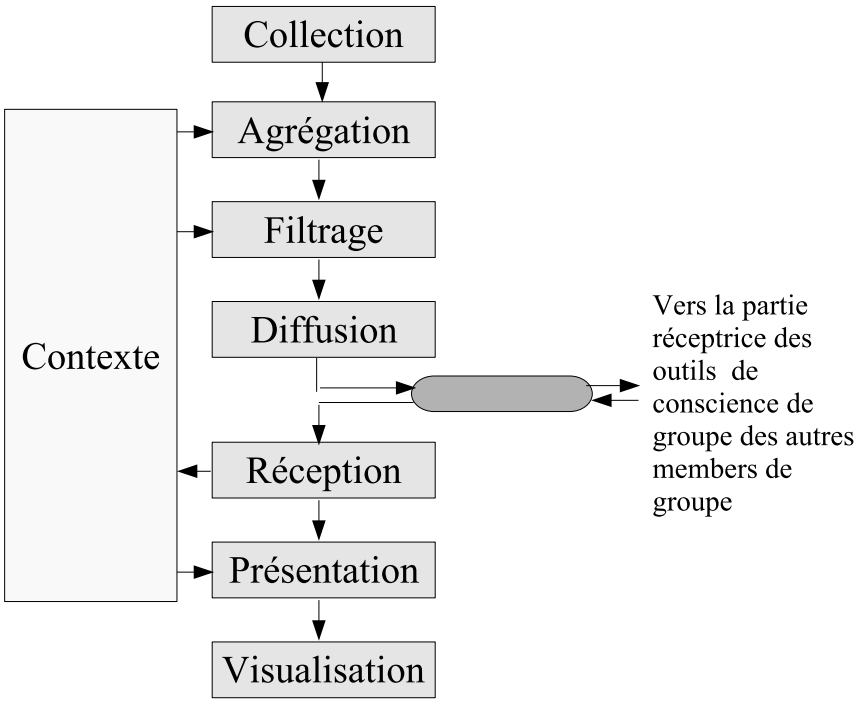

FIG. 3.1 – L'architecture fonctionnelle dans le P2P modèle [Bouthier, 2003]

- L'étape de filtrage : le contexte de travail est utilisé pour déterminer quelles informations doivent être diffusées et à qui.
- L'étape de l'adaptation : l'outil détermine le niveau d'intrusion avec lequel l'information doit être présentée à l'utilisateur selon son contexte de travail.

3.3 Une architecture fonctionnelle incluant un serveur

Nous proposons une extension de l'architecture fonctionnelle présentée ci-dessus pour intégrer un serveur offrant une gestion centralisée d'un certain nombre d'informations relatives au projet supportée. Notre architecture est répartie entre les clients et le serveur. La figure (3.2) montre comment les étapes de cette architecture sont réparties et comment ces parties interagissent.
Cet architecture a été construite en tenant compte des considérations suivantes :

- les informations disponibles sur le serveur sont *complémentaires* ce celles que l'on est capable de déduire sur les clients à partir de la modélisation probabiliste du comportement des utilisateurs. Le serveur dispose d'informations relatives à l'exécution d'activité prévues et formalisées dans le projet : tâches et leur enchaînement, structure organisationnelle, documents partagés. Le client quant à lui pourra obtenir des indications sur le contexte d'usage de l'utilisateur (lieu, instruments d'interaction) et son comportement (attention, activité réelle ...).
- le serveur produit beaucoup d'événements, qu'il est nécessaire d'agréger,
- l'adaptation au contexte individuel reste nécessaire et doit se faire sur chaque client,
- il faut essayer de minimiser les échanges entre clients et serveur. D'une part pour minimiser le traffic, et d'autre part parce que la connection entre le client et le serveur peut être périodique, si l'on prend en compte la possibilité de clients mobiles. Ainsi, on conserve

l'étape d'agrégation sur les clients.
- la mise en correspondance des informations et des destinataires est plus simple à faire sur le serveur, puisque celui-ci dispose d'informations plus complètes et formalisées sur les relations entre les membres du groupe : structure organisationnelle (par exemple, hiérarchie de responsabilité), liens liés à des dépendances d'activité (par exemple, un utilisateur attend le résultat d'un autre), documents ou ressources partagés.

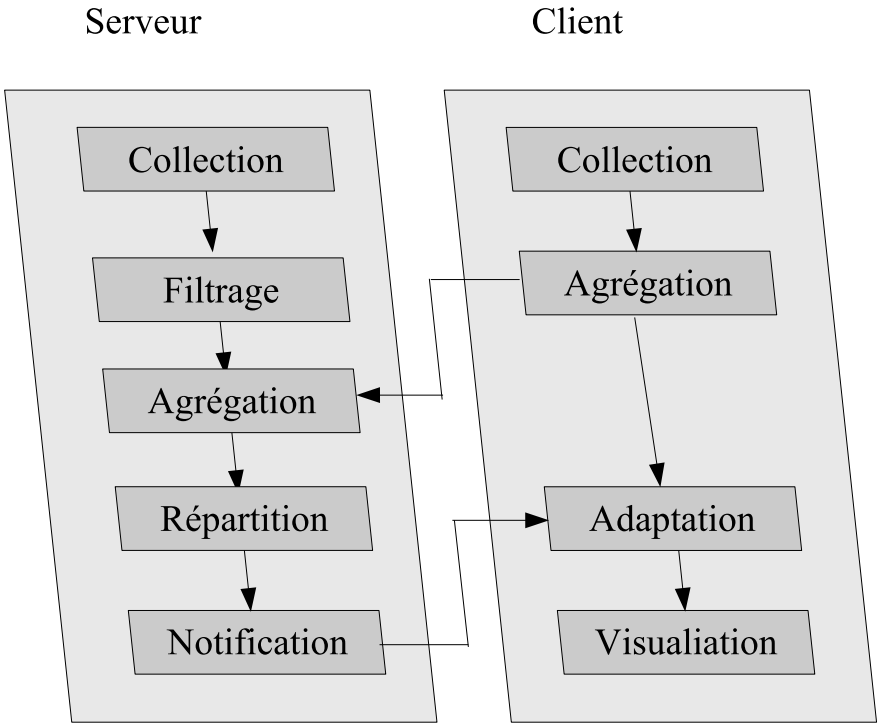

FIG. 3.2 – l'architecture fonctionnelle pour le client/server modèle

Les étapes proposées dans cette architecture sont détaillées dans la suite.

3.3.1 Collection

L'outil collecte des informations sur le serveur et sur le client. Sur le client l'outil collecte les informations importantes pour caractériser le contexte de travail individuel du client : applications courantes, fichier ouverts, les fréquence de frappe clavier ou de clicks souris, la date de chaque action etc...
Les informations sur le serveur sont de deux formes :
- Des informations persistantes et statiques définies à la configuration du projet, et évoluant peu : composition des groupes, hiérarchie, activités prévues et ressources affectées à ces activités.
- Des informations très dynamiques liées à l'exécution d'actions sur le serveur, caractérisées par des événements disponibles.

Dans LibreSource les informations statiques correspondent au contenu de l'arbre de ressources. Les informations dynamiques correspondent aux événements produits par les opérations exécutées sur les ressources. Ces événements sont collectés dans une queue persistante.

3.3.2 Filtrage

Dans le serveur beaucoup d'événements sont produits lors de la réalisation d'une activité complète. Cette étape de filtrage consiste à masquer les événements qui ne sont pas intéressants ou significatifs de façon à alléger l'étape d'agrégation.

3.3.3 Agrégation

Elle se passe sur le côté du serveur, et sur la côté du client. Sur le côté du client, l'agrégation est réalisée comme présentée précédemment, en utilisant des réseaux bayesiens. L'agrégation du côté serveur fait l'objet du chapitre suivant et utilise une technique différente.

3.3.4 Répartition

Cette étape, placée sur le serveur, détermine à qui le message de notification doit être envoyé et le niveau de détail des informations qui viennent de l'étape précédente.

Les critères qui affectent le processus de répartition sont les suivants :

- quand l'utilisateur crée une ressource, il est le propriétaire de cette ressource. Il peut donner des permissions d'accès ou d'exécutions d'opérations sur cette ressource aux autres utilisateurs, parce qu'il pense que ce membre du groupe va avoir besoin d'accéder a cette ressource : "I let you see what I want you to see". L'outil n'envoie pas de message de notification aux utilisateurs n'ayant aucune permission sur la ressource origine d'un événement.
- si l'utilisateur a déjà utilisé ce ressource plusieurs fois.
- la relation entre le client et l'utilisateur dans la hiérarchie de groupe et la liste de personnes proche du client. Normalement l'outil envoie aux personnes qui sont au même niveau. L'outil envoie aussi aux personnes proches du client.
- quelquefois on donne des priorités aux membres de passer tous les événements de ces membres à tous les membres de groupe comme le chef de groupe.
- Dans le service Web l'utilisateur peut déterminer par un filtrage quels sont les événements qui lui sont intéressants et il va recevoir seulement les événement selon ce filtre.

3.3.5 Notification

Après avoir déterminé les destinataires d'une information, l'étape de notification réalise l'acheminement effectif de l'information vers les clients. Cette étape détermine le mode d'acheminement à utiliser en fonction de l'état de l'interaction entre le client et le serveur.

3.3.6 Adaptation et Visualisation

Le rôle de ces étapes est inchangé par rapport à l'architecture de départ. Elle consiste à adapter la présentation des informations en fonction du contexte de travail individuel de chaque utilisateur.

Chapitre 4

L'agrégation sur le serveur

4.1 Introduction

Comme nous l'avons vu, dans LibreSource les événements qui se déclenchent sur le serveur sont nombreux. Nous avons mis en evidence une étape d'agrégation sur la partie serveur. Ce chapitre est consacré à la conception d'un mécanisme d'agrégation d'événements à mettre en oeuvre sur la partie serveur de l'architecture.

4.2 Choix de la technique d'agrégation

Pour concevoir le mécanisme d'agrégation d'événement sur le serveur, nous devons d'abord choisir une technique d'agrégation.

Sur la partie cliente, l'agrégation est faite en utilisant la technique proposée dans [Bouthier, 2003], basée sur l'utilisation de réseaux bayesiens. L'agrégation est réalisée au travers de plusieurs canaux, chacun ayant la responsabilité de répondre à une des questions suivantes :

- Quoi : quel est le type d'activité effectuée ?
- Qui : qui a fait cette activité ?
- Quand : quand cette activité s'est produite ?
- Où : où et sur quel ordinateur activité s'est produite ?
- Comment : quel sont les objets utilisée pour réaliser l'activité ?
- Pourquoi : dans quel cadre l'activité est-elle-effectuée ?

Les réponses des canaux sont agrégées ensemble dans une seule information haut niveau. L'utilisation du raisonnement probabiliste et des réseaux bayesiens se justifie par l'aspect incertain du lien reliant les événements observés à leurs causes possibles. Un événement pouvant avoir plusieurs causes, y compris des causes non identifiées, il est difficile de décider de sa cause réelle de façon sure.

Dans LibreSource, par contre, il n'y a aucune incertitude. Chaque événement à une cause unique et parfaitement identifiée puisque un événement est associé à une opération sur une ressource. Les propriétés de l'événement caractérisent de manière déterministe la ressource origine de l'événement.

Par ailleurs, lorsqu'un client demande l'execution d'une activité, cette execution peut conduire à la production de plusieurs événements sur différentes ressources. Par exemple pour l'activité "create workspace" les événements suivants se déclenchent sur le serveur :

```
- Kernal.chown
```

```
- Kernel.create
- Kernel.bind
- Kernel.createAcl
- LibreSource.workspace.connection.create
```

Actuellement, le serveur notifie les clients de ces événements en envoyant cinq messages, et c'est l'utilisateur qui doit agréger ces événements et déduire à partir d'eux qu'un espace de travail a été créé. Cependant cette séquence est connue d'avance : pour chaque activité, l'ensemble d'événements produit peut être identifié, et il sera toujours le même pour chaque execution de l'activité.

Nous pensons donc que l'utilisation d'une technique probabiliste à base de réseaux bayesiens n'est pas utile ni justifiée : une séquence d'événements observés permet de déterminer exactement, de manière certaine et unique, l'activité qui en est la cause et l'ensemble de ressources concernés. Le problème de l'agrégation consiste donc en la reconnaissance de séquences ou de motifs identifiés d'événements. Pour réaliser cette reconnaissance, nous proposons d'utiliser la technique des événements complexes (CEP) proposée par [Luckham, 2001].

4.3 Les événements complexes

Un événement complexe est une agrégation d'autre événements. Les événements membres d'un événement complexe peuvent être des événements qui se produisent à différents moments et dans des composants séparés dans le système. L'événement complexe est un événement de plus haut niveau que ses membres, qui permet d'avoir une vue abstraite des activités au sein d'un système. En d'autre terme, un événement complexe est une abstraction pour l'ensemble de ses événements membres.
Les relations entre les événements peuvent être :

1. Temporelle : l'événement A est avant l'événement B si A se produit avant B selon l'horloge du système.
2. Causalité : la relation entre A et B est causale si la production de l'événement A est nécessaire pour que l'événement B se produise.
3. Agrégation : Si l'événement A est une activité qui consiste en ensemble d'événements B1,B2,B3,... alors A est une agrégation de tout événement Bi. En revanche les événements Bi sont des membres de l'événement A.

4.3.1 Le Traitement des événements complexes

Un événement complexe est défini par une règle d'agrégation qui consiste en un motif à reconnaître. Un événement complexe est créé lorsque l'on observe un ensemble d'événements qui recouvre le motif défini dans la règle d'agrégation. Le mécanisme pour des événements complexes comprend deux composants :

- Les règles/motifs d'agrégation : un motif est défini par un ensemble d'événement et les relations entre ces événements : AND, OR , causale, parallèle et temporelle. La règle défini l'action qui va se produire quand le motif est recouvert.
- Des Agents de traitement des événements (Event Processing Agents - EPAs) : les Agents sont des objets qui exécutent les règles/motifs d'événements. Les agents sont les composants qui réalisent la reconnaissance de motifs et l'agrégation effective.

4.3.2 La hiérarchie d'abstraction des événements

La hiérarchie d'abstraction des événements est une structure logique qui fait le lien entre :
- les événements et activités effectivement observables dans le système (événements de bas niveau),
- les événements et activités que l'on souhaite observer dans le système (événements abstraits de haut niveau).

Cette hiérarchie permet de combler le fossé qui existe entre le niveau auquel on souhaite observer le comportment du système, et le niveau de production réel des événements. Pour construire une telle hiérarchie, il est nécessaire d'identifier

1. Le nombre de niveaux nécessaire : chaque niveau consiste en une description d'activités et des événements correspondant à ces activités. Le niveau 1, niveau le plus bas contient les événements produits directement par le système. Dans LibreSource, il s'agit d'événement JMS ("Java Message Service").
2. Un ensemble de règles d'agrégation : à chaque niveau, on définit des règles d'agrégation qui permettent d'obtenir les événements du niveau courant à partir des événements du niveau inférieur. Les règles d'agrégation sont ainsi des règles de transition de niveau dans la hiérarchie d'abstraction.

La figure (4.1) montre une hiérarchie d'abstraction d'événements (partielle) pour un ensemble d'événements dans LibreSource. Au niveau 1 sont listés les événements observables dans le système, et les activités qui les produisent. Au niveau 2, ces événements sont regroupés et mis en correspondance avec des activités de plus haut niveau. La figure (4.2) détaille la règle d'agrégation qui permet d'abstraire l'activité "Création d'une queue So6" à partir d'un ensemble d'événement du niveau 1.
L'annexe (A) donne une description de tous les événements dans LibreSource, ces événements forment le niveau 1 de notre hiérarchie. Le niveau 2 est composé des activités que l'on souhaite observer et notifier aux utilisateurs. Ses exemples sont donnés dans l'annexe (C).

Par ailleurs, [Luckham, 2001] propose un langage de description d'événement complexe, nommé RAPIDE EPL. Nous utiliserons ce langage pour exprimer des règles d'agrégation.

4.4 La mise en oeuvre

4.4.1 Les Agents de Traitement d'Evénements (EPA)

Un EPA est un objet qui réalise une (ou plusieurs) règle(s) d'agrégation. Il surveille un ensemble d'événements qui lui sont fournis en entrée afin de détecter certains motifs. Un EPA contient des règles basées sur des motifs et des variables locales dont les valeurs déterminent l'état de l'EPA. Chaque règle a deux parties :
- un motif d'événements,
- une action à déclencher lorsque le motif est recouvert.

L'agent exécute les actions lorsque la règle correspondante est activée. Le résultat de l'exécution d'une règle peut être un changement de valeur des variables locales ou la création d'un événement du niveau supérieur. Chaque EPA correspond à un seul thread de contrôle. Plusieurs EPAs peuvent s'exécuter parallèlement et communiquent en échangeant des événements.

Niveau	Activités	Types d'événement
Niveau 2	créer So6 queue	libresourceSynchronizer.synchronizer.create Kernel.create Kernel.bind libresourceFiles.file.download
	créer workspace	libresourceSynchronizer.workspace.connection.create Kernel.create Kernel.bind Kernel.create Kernel.chown
	...	...
Niveau 1	Les activités sur le service So6	libresourceSynchronizer.synchronizer.create libresourceSynchronizer.workspace.connection.create libresourceSynchronizer.synchronizer.edit libresourceSynchronizer.workspace.connection.edit libresourceSynchronizer.synchronizer.delete libresourceSynchronizer.workspace.connection.delete libresourceSynchronizer.workspace.connection.updateLastTic
	...	...

FIG. 4.1 – Hiérarchie (partielle) d'abstraction dans LibreSource

Element	Declaration
Variables	URL fromResource, Srting eventtype , String servicename URL throwedby, Date date, String args
Types d'événement	Kernel.bind Kernel.create libresourceSynchronizer.synchronizer.create libresourceFiles.file.download
Relation	AND
Pattern	Kernel.bindAND Kernel.create AND libresourceSynchronizer.synchronizer.create AND libresourceFiles.file.download
Action	Create So6 queue

FIG. 4.2 – Règle d'agrégation pour l'activité *création d'une queue So6*

4.4.2 La spécification des motifs

Le langage RAPIDE fournit un certains nombre d'opérateurs pour décrire les motifs d'agrégation. Nous en détaillons deux ici : les opérateurs temporels et les opérateurs de répétition.

Les opérateurs temporels

- At(T1) : le motif est recouvert par tout événement se produisant au temps T1,
- After(T1) : le motif est recouvert par tout événement se produisant à un temps supérieur au temps T1,
- During(T1,T2) : le motif est recouvert par tout événement se produisant entre les temps T1 et T2.

L'opérateur de répetition

L'orpérateur de répétition permet de spécifier des motifs comportant plusieurs occurences successives d'un même événement. La forme générale est la suivante :

```
[number of repetition  REL relational operator ] P;
```

où les *relationnel operator* spécifie une relation entre les événements répétés et P est l'événement qui se répète. Par exemple :

1. ```
   [*  REL -> ]  create;
   ```

   indique un motif constitué d'un nombre arbitraire d'occurrences de l'événement "Create", reliés entre elle par une relation causale,

2. ```
   [1..10 REL -> ]  create;
   ```

   indique un motif constitué de 10 occurrences de l'événement "Create", reliés entre elle par une relation causale,

3. ```
   [ I in 1..10 REL  ~ ]  create;
   ```

   indique un motif constitué d'un nombre compris entre 1 et 10 d'occurrences de l'événement "Create", reliés entre elle par une relation quelconque.

4.4.3 Les classes d'EPAs

Suivant leur rôle dans la hiérarchie d'abstraction, on distingue deux types (ou classes) principaux d'agents de traitement d'événements : les filtres et les Maps.

Les filtres utilisent un motif pour filtrer un ensemble d'événement. Le filtre laisse passer uniquement les événements satisfaisant le motif. La sortie d'un filtre est un sous-ensemble des événements en entrée : un filtre ne crée aucun événement. Un filtre est utilisé pour éliminer les événements non significatifs ou inutiles dans la caractérisation des activités de haut niveau. La figure (4.3) illustre la classe filtre : les événements A,B,C sont des événements entrants et A,C sont les événements sortants.

On peut filtrer sur différents types de conditions :
   ```

1. filtrage sur le nom des événements : la condition porte sur les noms des événements,
2. filtrage sur le contenu : la condition porte sur les attributs des événements,

FIG. 4.3 – interface of an event processing agent class MAP

Les Maps utilisent un motif d'événements pour agréger plusieurs événements en un seul de plus haut niveau. Cet agent est créateur de nouveaux événements. Lorsque le motif est reconnu, la règle crée un nouvel événement avec l'action **generate**. La figure (4.4) donne un exemple d'agent map. Cet agent reçoit trois types d'événements entrants A,B,C (son domaine) et produit l'événement seq(A,B,C) en sortie.

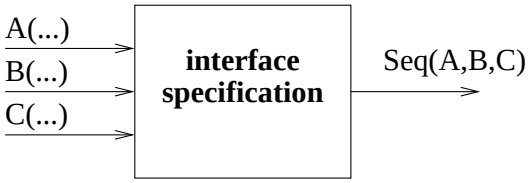

FIG. 4.4 – Interface of an event processing agent class MAP

4.4.4 Les réseaux de Traitement d'Événement (EPN)

Un réseau de traitement d'événement (EPN) est un ensemble d'agents EPA connectés. Il est représenté graphiquement comme un réseau dont les noeuds sont des agents EPAs et les arcs représentent la circulation d'événements dans le réseau. La construction d'un EPN permet de diviser le problème de reconnaissance d'événements complexe en étapes simples. Pour construire un EPN, on doit connecter des agents de types filtres ou maps de telle façon que la sortie d'un agent soit l'entrée de l'autre.
La figure (4.5) illustre la structure générale d'un EPN et la communication entre les différents composants. Les événements du système observé peuvent provenir de différents composants de ce système. Par exemple on peut observer des événements en provenance du réseau, du SGBD ou du middleware. Le rôle de la couche d'adaptation est d'uniformiser la représentation des événements traités par les couches supérieures. Le rôle de la couche filtrage est de réduire le nombre d'événements en restreignant aux seuls événement significatifs. Le rôle de la couche agrégation est de produire une vue abstraite des activités se déroulant dans le système.
Les différents agents d'un réseau peuvent être distribués sur plusieurs sites. Une organisation habituelle est de réaliser une agrégation locale sur chaque site puis d'agréger les vues des différents sites en une seule.

Les intérets de construire un EPN sont les suivants :

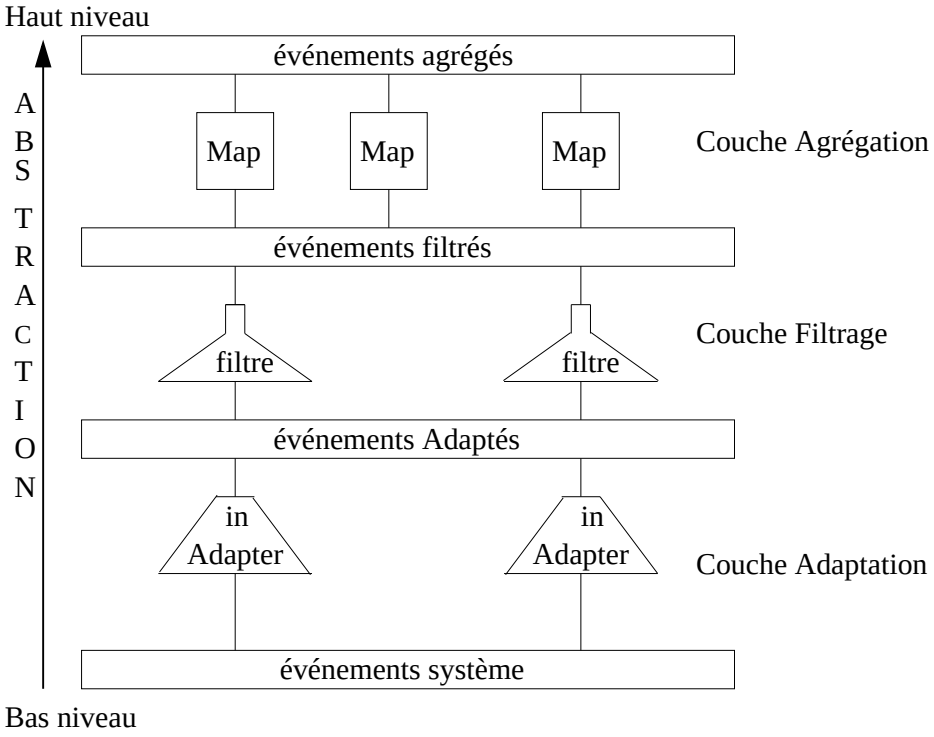

FIG. 4.5 – La structure de réseau de EPAs de [Luckham, 2001]

1. Analyse efficace d'événements : le système observé peut produire de nombreux événements. Ces événements sont filtrés le plus tôt possible pour éviter de surcharger les règles d'agrégation.
2. Flexibilité : on peut ajouter et supprimer des agents facilement dans l'EPN,
3. Présenter une vue globale de systèmes distribués.

Connexion entre les agents

Les connexions entre les agents associent les événements sortants d'un agant aux entrées d'un autre agent. Elles peuvent être de différents types :

1. Connexion séquentielle : les événements sont consommés l'un après l'autre, dans l'ordre où ils sont produits. La connexion conserve l'ordre.
2. Connexion parallèle : l'ordre de production n'est pas important pour la consomation des événements.
3. Connexion gardé : la connexion est soumise à la validation d'une condition logique portant sur les paramètres des événements en sortie. Les événements ne sont transmis à l'entrée connectés que si la condition est valide.

4.4.5 La Classe EPN

La classe EPN permet d'encapsuler un réseau de façon à le traiter comme un agent EPA. Ceci permet de réutiliser des réseaux complets et permet de connecter des réseaux à d'autres

Agents EPAs ou à d'autres réseaux EPNs. La figure (4.6) illustre la structure d'une classe EPN.

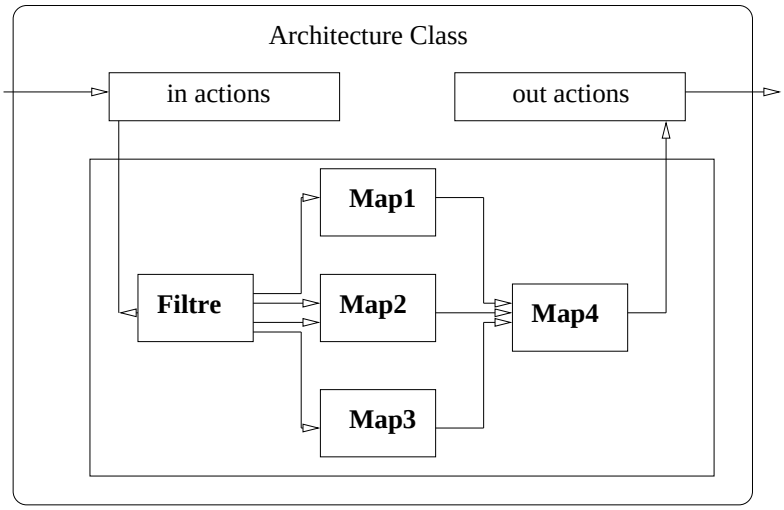

FIG. 4.6 – Exemple de classe EPN

4.5 Des événements complexes dans LibreSource

Nous proposons de réaliser le mécanisme d'agrégation d'événements dans le serveur LibreSource sur la base sur la base de l'approche décrite ci-dessus. La stratégie d'intégration de ce mécanisme dans la plateforme LibreSource est la suivante :

- Les événements de niveau 1 sont les événements produits par la plateforme LibreSource lors de l'exécution d'opération sur des ressources. Ces événements ont une représentation uniforme quelque-soit leur origine : ce sont des événements JMS. Nous conservons cette représentation, ce qui nous permet de nous passer de la couche d'adaptation (plus exactement, c'est en fait LibreSource qui réalise déjà l'adaptation).
- Comme tout service LibreSource, le service d'agrégation est fourni sous la forme de ressources. On utilisera deux types de ressources dans le service d'agrégation : les ressources de type *Filtre*, et les ressources de type *Map*. Chaque ressource créée correspondra à un agent de traitement d'événements.
- Comme toute ressource LibreSource, les ressources du service d'agrégation produisent des événements. Ces événements correspondent aux opérations effectuées sur la ressource (création, modification, suppression ...) plus les événements de sortie de l'agents de traitement implanté par la ressource.
- Un réseau de traitement d'événements (EPN) est défini par un arbre de ressources du service d'agrégation. La connexion entre les agents est implicitement définie par la hiérarchie de l'arbre : les événements produits par une ressource sont connectés à l'entrée de la ressource/agent immédiatement supérieure dans l'arbre. Les feuilles de l'arbre de ressources du service d'agrégation sont connectées aux queues d'événements persistants produisant les événements de base LibreSource. La racine de cet arbre produit des événements agrégés correspondant à une vue abstraite des activités se déroulant sur le serveur.

Les intérêts d'une telle approche sont multiples :

1. l'intégration est uniforme avec les autres services LibreSource et réutiliser les mécanismes existants. En particulier, un client souhaitant être notifié d'événements de haut niveau doit simplement souscrire à la queue d'événements associées à la ressource d'agrégation qu'il choisit. Le mécanisme de gestion des queues et de souscription est déjà présent dans la plateforme, et les clients n'ont pas à modifier leur comportement pour bénéficier de l'agrégation.
2. les événements produits par une ressource d'agrégation sont exploitables exactement comme tous les autres événements du système, au travers d'une queue d'événements associée à la ressource.
3. la hiérarchie d'abstraction n'est pas figée. Les ressources et les réseaux d'agrégation peuvent être créés et adaptés pour chaque projet, voir même par chaque client. On obtient donc une grande flexibilité et une grande facilité d'utilisation. Il est assez facile de rajouter des ressources dans une hiérarchie existante.
4. rien n'empêche de connecter des événements issus de plusieurs serveurs. Tous les serveurs présentant leurs événements de manière uniforme, on peut parfaitement envisager, même si nous ne l'avons pas fait, de connecter des événements issus d'un serveur à des ressources situés sur un autre serveur.

Nous définissons maintenant le processus de création des ressources dans ce cadre.

4.5.1 Définition des ressources d'agrégation

Dans LibreSource, le processus de création d'une ressource est un processus interactif qui se déroule dans une page Web. Le créateur de la ressource configure cette ressource en interagissant avec le serveur. Dans le cas des ressources du service d'agrégation, la création/configuration d'une ressource consiste à créer et paramétrer un agent de traitement d'événements. Les paramètres principaux sont : sa position dans le réseau de traitement, son type (filtre, map), les événements en entrée, le motif associé et les événements de sortie.

Ainsi, la création d'une ressource d'agrégation consiste à configurer les paramètres suivants :

1. Nom de la ressource : le nom de la ressource est un chemin dans l'arbre des ressources. Ce nom identifie la ressource et a également pour effet de déterminer sa position dans le réseau de reconnaissance. Les événements issus de cette ressource seront automatiquement fourni en entrée de la ressource mère dans l'arbre.
2. Type de la ressource : filtre ou map. Ce choix conditionne notamment les événements de sortie : sous-ensemble de l'entrée pour un filtre, événements nouveaux pour un Map.
3. Événements en entrée : de manière implicite, la position détermine dans l'arbre de ressources détermine l'ensemble d'événements fournis en entrée pour la ressource comme étant l'ensemble des événements de sortie des ressources filles. On peut cependant ajouter une spécification explicite en indiquant des types d'événements à consommer des les queues d'événements de LibreSource. Cette définition est utile pour :
   - le cas des feuilles de l'arbre,
   - les cas où on veut avoir accès à des événements produits par des ressources extérieures à l'arbre en cours de définition. C'est donc un moyen d'utiliser les événements de sortie d'une ressource d'agrégation dans plusieurs autres ressources d'agrégation. C'est aussi un moyen d'accéder à des événements produits par des ressources situées sur un autre serveur.
   - les cas où l'on veut injecter des événement de base à un niveau intermédiaire de la hierarchie.

4. Événements en sortie et règle d'agrégation : ces événements sont spécifiés par un motif de filtrage/map exprimé dans le langage RAPIDE. Une règle RAPIDE est entrée dans la ressource et sera interprétée par le code associé à la ressource. Cette règle doit indiquer les événements générés dans le cas d'un Map avec la clause **generate** de RAPIDE.

Un exemple est montré dans l'annexe (D).

4.5.2 Ressources d'agrégation par défaut et prédéfinies

Le processus de création d'un réseau complet de traitement des événements peut être complexe et difficile. Pour alléger cette tâche nous proposons deux solutions complémentaires : l'existence de ressources d'agrégation par défaut et la possibilité de prédéfinir des ressources d'agrégation pour une installation de LibreSource.

Les ressources d'agrégation par défaut sont des ressources automatiquement utilisées par le système si un client donné n'a accès ou n'a créé aucune ressource d'agrégation dans son environnement.
Ces ressources par défaut réalisent l'agrégation d'événements correspondant à quelques activités habituelles et caractéristiques d'une utilisation classique de la plateforme LibreSource :

- création d'un projet,
- création d'un espace de travail,
- création d'une queue de synchronisation,
- démarrage des activités prévues dans le modèle de workflow ...

La liste des agrégations par défaut est ouverte. Nous avons identifié et décrit les principales, mais il est relativement aisé d'en rajouter.

Les ressources d'agrégation prédéfinies sont des ressources définies par l'administrateur d'une installation LibreSource, qui les mets à disposition des utilisateurs de la plateforme, qui peuvent les réutiliser directement sans avoir à les reconstruire eux-même. La différence avec les ressources par défaut est que ces ressources ne sont automatiquement actives, mais doivent être explicitement réutilisées par le client qui veut en profiter.

Ces 2 types de ressources sont fournies dans le système d'une manière analogue : il s'agit d'un réseau de traitement encapsule dans une ressource d'agrégation de type EPN. Elles apparaissent donc toujours comme une ressource unique, même si elles correspondent à un réseau complexe.

Un exemple pour faire l'agrégateur pour SO6 et Files services, la classe pour So6 est montrée en l'annexe (E)

4.6 La réalisation

Le cadre de réalisation de ce travail est la plateforme LibreSource et la langage de programmation est Java 2 enterprise Edition (J2EE). LibreSource est déployé sur le serveur JEE JOnAS serveur. JOnAS est un serveur d'application pur Java, open source, se conformant à la spécification J2EE, construit par le consortium français ObjectWeb.

Nous avons créé un nouveau service LibreSource et implanté les ressources associées. Pour l'instant, seule la ressource de type Filtre a été finalisée. Ce services contient 2 modules : LibreSourceAggregator qui conteient les ressources de types Map, et LibreSourceFilter qui contient

les ressources de type Filtre. Les actions possibles sur ces ressources sont les actions cimmunes à toutes les ressources :

```
Module Name Service Ressources           Action aggregator
           LibreSourceAggregator  aggregator  create
           LibreSourceAggregator  aggregator  edit
           LibreSourceAggregator  aggregator  delete

           LibreSourceFiltrage    filtrage    create
           LibreSourceFiltrage    filtrage    edit
           LibreSourceFiltrage    filtrage    delete
```

L'implantation d'une ressource revient à réaliser un composant particulier de conformant à une interface prédéfinie commune à toutes les ressources. La conformation à cette interface permet l'intégration de la nouvelle ressource dans la plateforme. Il faut aussi réaliser l'interface de configuration de la ressource qui permet de fixer les paramètres de la ressources lors de sa création.
Du point de vue de la réalisation, le point difficile est la construction de la machine d'interprétation des motifs RAPIDE. Nous avons seulement réalisé la partie consacré aux motifs de filtrage.

Conclusion et perspectives

Ce mémoire présente une approche pour l'agrégation des événements de conscience de groupe dans une plateforme de coopération basée sur un serveur. Nous avons proposé dans un premier temps une architecture fonctionnelle inspirée de travaux précédents. Notre proposition a pour objectif de réutiliser et de s'intégrer au mieux avec les propositions déjà faites dans [Bouthier, 2003], tout en prenant en compte la spécificité d'une architecture client/serveur par rapport à une architecture purement pair-à-pair. Même si nous n'avons pas encore pu tester l'intégration effective des 2 approches, nous pensons que notre approche est cohérente.

Dans un deuxième temps, nous avons proposé une méthode d'agrégation des événements sur le serveur, ainsi que sa stratégie d'intégration dans la plateforme LibreSource. Cette méthode est basée sur la notion de traitement d'événements complexe (CEP) proposée par [Luckham, 2001]. Nous pensons que cette stratégie est mieux adaptée aux besoins d'agrégation sur le serveur que celle qui a été proposée dans [Bouthier, 2003]. Enfin l'intégration dans la plateforme LibreSource a été relativement simple à partir du moment où nous avons choisi de réaliser l'agrégateur sous la forme de ressources comme les autres services.

Il reste beaucoup de travail à réaliser, en particulier du point de vue de la réalisation. Pour l'instant, seule les ressources pour le filtrage ont été réalisées (et pas complètement testées).

Notre première perspective concerne donc la réalisation complète du service d'agrégation dans LibreSource. Un des intérêts du service tel qu'il a été conçu est qu'il peut être utilisé tel quel, sans forcément être complété par les autres étapes de l'architecture fonctionnelle proposée dans le chapitre 3.

Notre perspective à moyen terme consiste à réaliser les autres étapes de cette architecture fonctionnelle que nous avons proposé. Si l'agrégation et l'adaptation sur le client peuvent être réutilisée à partir des propositions de [Bouthier, 2003] l'étape de répartition doit être revue pour exploiter au mieux les informations et connaissances disponibles sur le serveur à propos des relations entre les participants d'un projet.

Enfin, notre objectif à long terme est de construire, sur la base de cette architecture, un système de conscience de groupe continue, c'est à dire capable de suivre l'utilisateur dans différents contextes d'usage, et de s'adapter à chaque contexte. Ceci nécessitera de prendre en compte une représentation plus large du contexte de travail, en y incluant notamment des informations en provenance de l'environnement physique : lieu (bureau, salle de réunion, voiture ...), instrument d'interaction (ordinateur de bureau, PDA, téléphone) ...

Annexe A

Les événements dans LibreSource

```
Module Name Service Ressources  Action

Kernel      kernel  (node)  create
            kernel  (node)  delete
            kernel  (node)  deleteURI
            kernel  (node)  bind
            kernel  (node)  unbind
            kernel  (node)  moveFrom
            kernel  (node)  moveTo
            kernel  (node)  chown
            kernel  (node)  createAcl
            kernel  (node)  deleteAcl
            kernel  (node)  resetAcls
            kernel  (node)  setProperty

            membership  user    edit
            membership  user    delete
            membership  group   create
            membership  group   edit
            membership  group   delete
            membership  group   addMember
            membership  group   removeMember


Core        LibreSourceCore symbolicLink    create
            LibreSourceCore symbolicLink    edit
            LibreSourceCore template    create
            LibreSourceCore template    edit
            LibreSourceCore timeline    create
            LibreSourceCore timeline    edit
            LibreSourceCore project create
            LibreSourceCore project edit
            LibreSourceCore project editSummary
```

```
BugTracker  LibreSourceBugTracker   bugTracker  create
            LibreSourceBugTracker   bugTracker  edit
            LibreSourceBugTracker   BugTracker  delete
            LibreSourceBugTracker   issue   create
            LibreSourceBugTracker   issue   edit
            LibreSourceBugTracker   issue   delete
            LibreSourceBugTracker   issue   assign
            LibreSourceBugTracker   issue   resolve


Files       LibreSourceFiles    file    create
            LibreSourceFiles    file    edit
            LibreSourceFiles    file    delete
            LibreSourceFiles    file    download
            LibreSourceFiles    repository  create
            LibreSourceFiles    repository  edit
            LibreSourceFiles    repository  delete


Forum       LibreSourceForum    message create
            LibreSourceForum    message edit
            LibreSourceForum    message delete
            LibreSourceForum    thread  create
            LibreSourceForum    thread  edit
            LibreSourceForum    thread  delete
            LibreSourceForum    forum   create
            LibreSourceForum    forum   edit
            LibreSourceForum    forum   delete


Mailing     LibreSourceMailing  mailingList create
            LibreSourceMailing  mailingList edit
            LibreSourceMailing  mailingList delete
            LibreSourceMailing  mailingList subscribe
            LibreSourceMailing  mailingList unsubscribe
            LibreSourceMailing  mailingList sendMessage


So6         LibreSourceSynchronizer synchronizer    create
            LibreSourceSynchronizer synchronizer    edit
            LibreSourceSynchronizer synchronizer    delete
            LibreSourceSynchronizer workspace   connection  create
            LibreSourceSynchronizer workspace   connection  edit
            LibreSourceSynchronizer workspace   connection  updateLastTicket
            LibreSourceSynchronizer workspace   connection  delete
            LibreSourceSynchronizer synchronizer    getPatch
            LibreSourceSynchronizer synchronizer    addPatch
```

```
            LibreSourceSynchronizer synchronizer    editPatch
            LibreSourceSynchronizer synchronizer    removePatch


Survey      LibreSourceSurvey   survey  create
            LibreSourceSurvey   survey  edit
            LibreSourceSurvey   survey  delete
            LibreSourceSurvey   survey  addOption
            LibreSourceSurvey   survey  deleteOption
            LibreSourceSurvey   survey  vote
            LibreSourceSurvey   survey  close
            LibreSourceSurvey   survey  open


Wiki        LibreSourceWiki page    create
            LibreSourceWiki page    editContent
            LibreSourceWiki page    delete
```

Annexe B

Les événements dans LibreSource qui passent du filtre

Module Name	Service Ressources	Action
Kernel	kernel (node)	delete
BugTracker	LibreSourceBugTracker	bugTracker create
	LibreSourceBugTracker	bugTracker edit
	LibreSourceBugTracker	issue create
	LibreSourceBugTracker	issue edit
	LibreSourceBugTracker	issue assign
	LibreSourceBugTracker	issue resolve
Files	LibreSourceFiles	file create
	LibreSourceFiles	file edit
	LibreSourceFiles	file download
	LibreSourceFiles	repository create
	LibreSourceFiles	repository edit
Forum	LibreSourceForum	message create
	LibreSourceForum	message edit
	LibreSourceForum	thread create
	LibreSourceForum	thread edit
	LibreSourceForum	forum create
	LibreSourceForum	forum edit
Mailing	LibreSourceMailing	mailingList create
	LibreSourceMailing	mailingList edit
	LibreSourceMailing	mailingList subscribe
	LibreSourceMailing	mailingList unsubscribe
	LibreSourceMailing	mailingList sendMessage
So6	LibreSourceSynchronizer	synchronizer create
	LibreSourceSynchronizer	synchronizer edit
	LibreSourceSynchronizer	workspace connection create

```
            LibreSourceSynchronizer workspace   connection  edit
            LibreSourceSynchronizer workspace   connection  updateLastTicket

Survey      LibreSourceSurvey   survey  create
            LibreSourceSurvey   survey  edit
            LibreSourceSurvey   survey  addOption
            LibreSourceSurvey   survey  deleteOption
            LibreSourceSurvey   survey  vote
            LibreSourceSurvey   survey  close
            LibreSourceSurvey   survey  open


Wiki        LibreSourceWiki page    create
            LibreSourceWiki page    editContent
```

Annexe C

Les événements dans LibreSource au niveau 2 dans la hiérarchie d'Événement

activité	les événements
craete So6 queue	
	kernel.create
	Kernel.bind
	LibreSourceSynchronizer.synchronizer.create
	LibreSourceFiles.file.download
create workspace	
	kernel.create
	Kernel.bind
	kernel.chown
	kernel.createAcl
	LibreSourceSynchronizer.workspace.connection.create
commit	
	LibreSourceSynchronizer.workspace.connection.updateLastTick
edit project	
	LibreSourceCore.project.edit
	LibreSourceFiles.file.download
create forum	
	kernel.create
	kernel.bind
	LibreSourceForum.forum.create

Annexe D

Un exemple pour définir un filtre et un Map

On définit le filtre dans LibreSource en utilisant RAPIDE. Ce filtre est de type filtre nom, qu'il vas passer les événements selon le nom.

une liste d'événements dans libresourec contient les événement suivants :

```
- LibreSourceSynchronizer.synchronizer.create
- LibreSourceSynchronizer.synchronizer.edit
- kernel.bind
- kernel.create
```

ces événements sont des événements entrants à ce filtre, et les événements sortants de ce filtre sont :

```
- LibreSourceSynchronizer.synchronizer.create
- LibreSourceSynchronizer.synchronizer.edit
```

on définit le filtre :

```
FILTRE  Filtre1 IN  ToutEvent OUT EventFiltre
{
          LibreSourceSynchronizer.synchronizer.create => PASS
          LibreSourceSynchronizer.synchronizer.edit   => PASS
}
```

où *ToutEvent* et *EventFiltre* sont des executions types.
les événements qui vont passer sont dans l'exécution type *EventFiltre*.
on définit *ToutEvent* d'abord :

```
 TYPED EXECUTION {
          LibreSourceSynchronizer.synchronizer.create
          LibreSourceSynchronizer.synchronizer.edit
          kernel.bind
          kernel.create
} ToutEvent
```

on définit aussi l'exection type *EventFiltre*
les événements dans EventFiltré doit être sous ensemble de ToutEvent.

```
TYPED EXECUTION {
        LibreSourceSynchronizer.synchronizer.create
        LibreSourceSynchronizer.synchronizer.edit
} EventFiltre
```

Un agrégateur dans LibreSource est une ressource et dans la page de création d'agrégateur les critères que le client va les déterminer à son choix.

si le client choisit de faire l'agrégation des événements qui a le nom create de telle façon que le nombre de cet événement soit 3, le pattern est :

```
[3 REL ~] create
```

La définition de map agent est :

```
MAP CreateAgreger IN In_Action OUT Out_Action
{
        [3 REL ~] create
==> GENERATE  SeqEvent
}
```

où In_Action et Out_Action sont des executions types. Il peut aussi déterminer l'agrégation selon d'un interval de temps.

```
MAP CreateAgreger1 IN InAction OUT OutAction
{
        create(service,time,user,...) AFTER 12:00
==> GENERATE SeqEvent(...)
}
```

Dans cet exemple l'agent *CreateAgreger1* agrège les événements "create" qui se déclenchent après 12h00. aussi le client peut déterminer l'agrégation selon qui a exécuter l'événement.

exemple

```
MAP CreateAgreger2 IN InAction OUT OutAction
{
        create(service,time,user...) WHERE user=client1
==> GENERATE SeqEvent
}
```

Dans cet exemple l'agent *CreateAgreger2* agrège les événements qui s'est fait par user qui a le nom *client1*.

Annexe E

un exemple pour faire l'agrégateur pour SO6 et Files services, la classe pour So6

Ici, un exemple pour faire l'agrégateur pour SO6 et Files services, la classe pour So6 est montrée en figure(E.1) la critère de faire l'agrégation est au choix de l'administrateur.

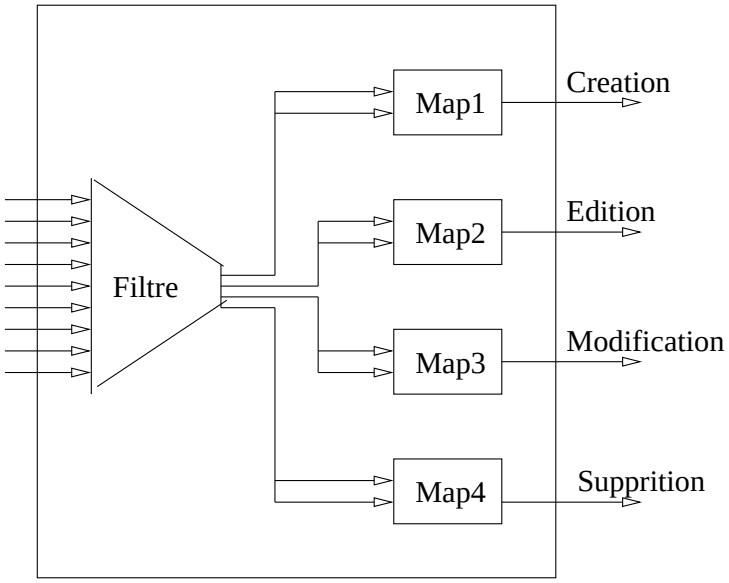

FIG. E.1 – L'architecture classe pour le merger agent

l'architecture classe pour le service So6
La procédure pour faire l'agrégation de cette classe sont :

1. déterminer les événements qui entrent et qui sortent de chaque classe.
   les entrées de la classe de So6 sont :

   - `LibreSourceSynchronizer.synchronizer.create`
   - `LibreSourceSynchronizer.synchronizer.edit`
   - `LibreSourceSynchronizer.synchronizer.delete`
   - `LibreSourceSynchronizer.workspace.connection.create`
   - `LibreSourceSynchronizer.workspace.connection.edit`

```
- LibreSourceSynchronizer.workspace.connection.updateLastTicket
- LibreSourceSynchronizer.workspace.connection.delete
- LibreSourceSynchronizer.synchronizer.getPatch
- LibreSourceSynchronizer.synchronizer.addPatch
- LibreSourceSynchronizer.synchronizer.editPatch
- LibreSourceSynchronizer.synchronizer.removePatch
```

les complexes événements sortants après avoir fait l'agrégation sont :

```
- création de so6 queue et workspace.
- edition de so6 queue et workspace.
- suppression de so6 queue et workspace.
- modification sur le workspace.
```

2. déterminer les événements qui passent dans le filtre

```
- kernel.delete
- LibreSourceSynchronizer.synchronizer.create
- LibreSourceSynchronizer.synchronizer.edit
- LibreSourceSynchronizer.workspace.connection.create
- LibreSourceSynchronizer.workspace.connection.edit
- LibreSourceSynchronizer.workspace.connection.updateLastTicket
```

3. définir des EPAs de type map (agrégateur) pour chaque événement sortant, déterminer le pattern par lequel l'EPA agrège les événements. par exemple le Map pour faire l'événement complexe "création de so6 queue et workspace" vient d'agréger les événements :

```
- LibreSourceSynchronizer.synchronizer.create
- LibreSourceSynchronizer.workspace.connection.create
```

L'architecture classe pour File est identique que celle pour So6 et pour faire l'agent merger on définit un autre Map de type agrégateur qui prend comme entrée les événements sortants des architectures classes de So6 et Files. La figure (E.2) montre l'architecture du merger agent. L'agent merge prend comme entrées les événements sortant de n'importe quelle architecture classe, l'administrateur a le choix de déterminer pour quel service l'agent merger appliquera.

E.0.1 Filtre et agrégateur prédéfinit

L'administrateur peut faire des filtres et des agrégateurs selon des critères générales et les clients peuvent souscrire à ces agrégateurs et recevoir les événements agrégés selon ces agrégateurs.

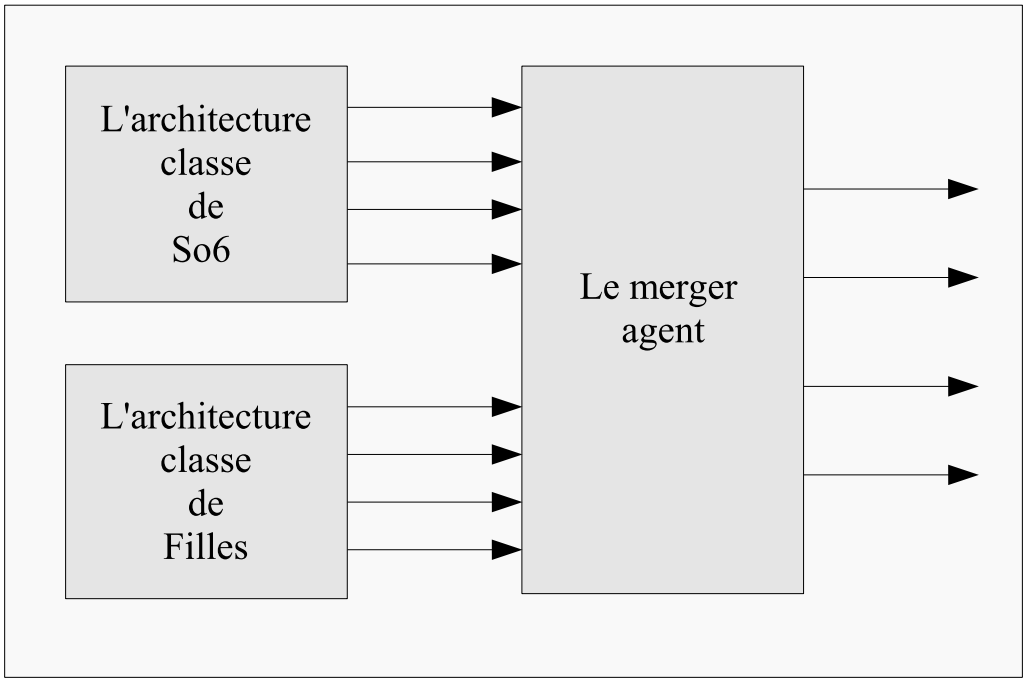

FIG. E.2 – EPN pour So6

Bibliographie

[Rev, 2003] (2003). *Review and Analysis of Event Notification Services.*

[Lib, 2005] (2005). Projet libresource.

[sap, 2005] (2005). sapdesignguild. In *http ://www.sapdesignguild.org/editions/edition5/glossary.asp.*

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *HUC '99 : Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag.

[Adi and Etzion, 2004] Adi, A. and Etzion, O. (2004). Amit - the situation manager. *VLDB J.*, 13(2) :177–203.

[Bentley et al., 1995] Bentley, R., Horstmann, T., Sikkel, K., and Trevor, J. (1995). Supporting collaborative information sharing with the {WWW} : The {BSCW} shared workspace system. pages 63–74.

[Bouthier, 2003] Bouthier, C. (2003). *Mise en contexte de la conscience de groupe : Adaptation et visualisation.* Thèse d'informatique, Université Henri Poincaré, Nancy1.

[Charniak, 1991] Charniak, E. (1991). Bayesian networks without tears : making bayesian networks more accessible to the probabilistically unsophisticated. *AI Mag.*, 12(4) :50–63.

[Dourish and Belloti, 1992] Dourish, P. and Belloti, V. (1992). Awareness and coordination in shared workspaces. In *Proceedings of the ACM CSCW Conference (CSCW-92)*, Toronto, Ontario. ACM Press.

[Fitzpatrick et al., 1999] Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T., and Segall, B. (1999). Augmenting the workaday world with elvin. In *Proceedings of the Sixth European conference on Computer supported cooperative work*, pages 431–450. Kluwer Academic Publishers.

[Fuchs, 1997] Fuchs, B. (1997). *Représentation des connaissances pour le raisonnement à partir de cas.* Thèse d'informatique, l'université Jean Monnet de saint-Etienne.

[Gross and Prinz, 2003] Gross, T. and Prinz, W. (2003). Awareness in context : a light-weight approach. In *ECSCW 2003 : Proceedings of the Eighth European Conference on computer supported cooperative work*, pages 259–341. kluwer academic publishers.

[Gutwin and Greenberg, 1996] Gutwin, C. and Greenberg, S. (1996). Workspace awareness for groupware. In *CHI '96 : Conference companion on Human factors in computing systems*, pages 208–209. ACM Press.

[Gutwin and Greenberg, 2002] Gutwin, C. and Greenberg, S. (2002). A descriptive framework of workspace awareness for real-time groupware. *Comput. Supported Coop. Work*, 11(3) :411–446.

[Gutwin et al., 1996] Gutwin, C., Rosman, M., and Greenberg, S. (1996). A usability study of awareness widgets in a shared workspace groupware system. In *Proceeding of the ACM CSCW Conference (CSCW-96)*, Cambridge, MA. ACM Press.

[Horvitz et al., 1998] Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. (1998). The lumiere project : Bayesian user modeling for inferring the goals and needs of software users.

[Luckham, 2001] Luckham, D. (2001). *The Power of Events : An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc.

[luckham, 2002] luckham, D. (2002). Draft paper achieving instant insight into the real-time electronic enterprise.

[Nomura et al., 1998] Nomura, T., Hayashi, K., Hazama, T., and Gudmundson, S. (1998). Intercolus : Mecanismes de configuration de zone de travail pour la consience d'activité. 12(4) :50–63.

[Prinz, 1999] Prinz, W. (1999). Nessie : an awareness environment for cooperative settings. In *Proceedings of the Sixth European conference on Computer supported cooperative work*, pages 391–410. Kluwer Academic Publishers.

[Rodden, 1996] Rodden, T. (1996). Populating the application : a model of awareness for cooperative aplpications. In *Proceedings of the ACM CSCW Conference (CSCW-96)*, Cambridge, MA. ACM Press.

[Steinfield et al., 1999] Steinfield, C., C., J., and Pfaff, P. (1999). Supporting virtual team collaboration : the teamscope system. In *GROUP'99 : Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 81–90, Phoenix, Arizona, United States. ACM Press.

[Tam and Greenberg, 2004] Tam, T. and Greenberg, T. (2004). A framework for asynchronous change awareness in collaboratively-constructed documents. In *CRIWG*, pages 67–83.