CoLab : conception et mise en oeuvre d'un outil pour la navigation coopérative sur le web

Guillermo de Jesus Hoyos Rivera

Année 2005

THÈSE

Préparée au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

En vue de l'obtention du
Doctorat de l'Université Paul Sabatier

Spécialité : *Informatique*

par
Guillermo de Jesús HOYOS RIVERA

CoLab : Conception et Mise en Œuvre d'un Outil pour la Navigation Coopérative sur le Web

---

Soutenue le 10 juin 2005 devant le jury :

Président	G. JUANOLE
Directeur de Thèse	J.P. COURTIAT
Rapporteurs	K. BARKAOUI E. HORLAIT
Examinateurs	M. DIAZ H. GUYENNET

Rapport LAAS N°

À Susy et à mes deux filles, Susita et Fabita

À mes parents, « papá Memo » et « mamá Panchis »

À ma « tía Carmela » et à mon « tío Rafael » (*In Memoriam*)

À Fabián Hernández

À Aldo Esteban Hernández Montiel (*In Memoriam*)

À toute ma famille et tous mes amis

À la vie…

"« Notre langue » ! Par-dessus les pages que lisait notre grande-mère, nous nous regardâmes, ma sœur et moi, frappés d'une même illumination : « … qui n'est pas pour vous une langue étrangère ». C'était donc cela, la clef de notre Atlantide ! La langue, cette mystérieuse matière invisible et omniprésente, qui atteignait par son essence sonore chaque recoin de l'univers que nous étions en train d'explorer. Cette langue qui modelait les hommes, sculptait les objets, ruisselait en vers, rugissait dans les rues envahies par les foules, faisait sourire une jeune tsarine venue du bout du monde… Mais surtout, elle palpitait en nous, telle une greffe fabuleuse dans nos cœurs, couverte déjà de feuilles et de fleurs, portant en elle le fruit de toute une civilisation. Oui, cette greffe, le français."

Extrait de « Le testament français »
Andreï MAKINE
Gallimard, Collection Folio, 1997
ISBN : 2070401871

Avant propos

Les travaux présentés dans cette thèse sont le résultat de recherches réalisées dans le Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS), dirigé, au cours de mon séjour par M. Jean-Claude LAPRIE puis par M. Malik GHALLAB, que je tiens à remercier cordialement pour leur accueil.

Je tiens à remercier M. Michel DIAZ, directeur de recherche au CNRS et responsable du groupe Outils et Logiciels pour la Communication (OLC) au moment de mon arrivée au LAAS-CNRS, de m'avoir accueilli au sein de son groupe de recherche.

Je tiens à remercier le CONACyT (Conseil National de Science et Technologie), qui a financé partiellement mon séjour doctoral en France à travers la bourse numéro 70360, ainsi que l'Universidad Veracruzana, qui m'a soutenu à travers une bourse complémentaire PROMEP. Je remercie ainsi tous les millions de mexicains, qui à travers le CONACyT et l'Universidad Veracruzana, ont financé mon séjour en France. Je ferai de mon mieux pour vous rétribuer de tout ce que j'ai reçu.

Je voudrais remercier très spécialement mon Directeur de Recherche, M. Jean-Pierre COURTIAT, pour avoir été beaucoup plus qu'un directeur de recherche. Merci, Jean-Pierre, de tous tes conseils, commentaires, recommendations, et de tout l'engagement et le temps que tu as passés avec moi pour m'aider à finaliser ce travail. Merci encore, de manière plus personnelle, pour ton amitié, ainsi que celle de ta belle famille. Vous êtes tous devenus une famille pour nous.

Je voudrais également exprimer mes plus vifs remerciements aux membres du jury :

- M. Guy JUANOLE, Professeur à l'Université Paul Sabatier, Toulouse
- M. Kamel BARKAOUI, Professeur au Conservatoire National des Arts et Métiers, Paris
- M. Eric HORLAIT, Professeur au LIP6, Paris
- M. Michel DIAZ, Directeur de Recherche CNRS, Toulouse
- M. Hervé GUYENNET, Professeur à l'UFR Sciences et Techniques, Besançon
- M. Jean-Pierre COURTIAT, Directeur de Recherche CNRS, Toulouse

pour l'honneur qu'ils m'ont fait d'accepter d'examiner mon travail de thèse, et particulièrement M. Kamel BARKAOUI et M. Eric HORLAIT pour avoir accepté la charge de rapporter sur cette thèse. De même, je voudrais remercier M. Guy JUANOLE de m'avoir fait l'honneur de présider ce jury.

Je ne peux pas oublier l'ensemble des membres du groupe OLC, aussi bien permanents que thésards, qui m'ont toujours soutenu, en particulier Mme. Véronique BAUDIN-THOMAS et M. Thierry VILLEMUR, M. Bernard BETHOMIEU et M. Pierre-Olivier RIBET. Je remercie également les services administratifs et logistiques du LAAS, en particulier Mme. Anne BERGEZ, M. Christian BERTY, Mme. Gina BRIAND, Mme. Régine DURAN et Mme. Noëlle ROUSSEL.

Je suis très reconnaissant également envers M. Pierre de SAQUI-SANNES, qui m'a beaucoup aidé et conseillé dans le developpement du modèle en UML du système de navigation coopérative.

Je remercie également tous mes amis au LAAS et à Toulouse, en particulier Roberta LIMA-GOMES, David Rafael GARDUÑO-BARRERA, José Valentim DOS SANTOS FILHO, Laurent GONZALEZ et Anne-Marie SOLACOUP et leurs enfants, Sara MOTA-GONZÄLEZ, Francisco MOO-MENA, Cristina ARZÁPALO-CENTENO, Guillermo ROMO-GUZMÁN et sa famille, Guillermo CORTÉS-ROBLES et sa famille, Saul POMARES-HERNÁNDEZ, Antonio MARÍN-HERNÁNDEZ, et tous les autres avec qui j'ai passé de très beaux moments, et qui m'ont toujours soutenu.

Mes plus grands remerciements spécialement à Roberta LIMA-GOMES, qui m'a beaucoup soutenu et aidé dans le developpement de mon travail, aussi bien au niveau conceptuel qu'au niveau d'implémentation. Merci aussi Roberta de ton amitié, et de ton esprit de travail en équipe.

Un remerciement spécial aussi à mon ami David Rafael GARDUÑO-BARRERA, avec qui j'ai partagé beaucoup de beaux moments, et qui m'a également aidé dans le developpement de mon travail. Merci pour toutes ces heures de discussion, et d'avoir été à mes côtés lors de la création de MexTlse. Merci aussi pour tes conseils et commentaires, aussi bien dans le cadre professionnel que personnel.

Au Mexique, je voudrais remercier mes parents, « papá Memo » et « mamá Panchis », pour m'avoir appris ce que veut dire être quelqu'un de bien. Merci de votre temps, de votre dedication en tant que parents, et d'avoir tant insisté que je revienne à l'école après l'avoir quitté.

Je veux aussi remercier ma « tía Carmela » et mon « tío Rafael » (†), qui ont toujours été aux côtés de ma famille, spécialement dans les temps difficiles. Il n'y a aucun doute que vous avez représenté un grand soutien pour nous tous.

Merci aussi à la « tía Lucha » et au « tio Mario », qui nous ont toujours soutenu et conseillé, aussi bien dans le cadre familial que professionnel. Vous êtes un exemple à suivre.

Mon souvenir et remerciement vont aussi à Fabián HERNANDEZ, qui m'a conseillé de revenir à l'Université pour continuer mes études en Informatique. Fabián, si tu n'avais pas fait ce petit mais très pertinent commentaire ce jour-là, je ne serais pas fort probablement revenu à l'école, et je ne serais jamais arrivé même à rêver à des objectifs tel que celui qui se realise aujourd'hui.

Merci aussi à la Maestra Virginia Angélica GARCIA-VEGA, pour m'avoir accordé sa confiance et avoir refusé ma démission dans ces temps difficiles quand je commençais le Mastère. Je t'en serais toujours reconnaissant.

Mes pensées viennent aussi à Clementina GUERRERO (Tula), qui m'a accordé sa confiance et soutenu au moment crucial de la finalisation de mes études de Mastère en Intelligence Artificielle, et le debut de mon metier de chercheur.

Ce travail est spécialement dédié à la mémoire de mon cher ami, Aldo Esteban HERNÁNDEZ-MONTIEL (†). Tu seras toujours dans mes pensées. Je suis sûr qu'un jour nous pourrons « echar lengua » de nouveau.

Pour terminer, je veux remercier Susy, Susita et Fabita, ma petite famille. Merci de votre soutien et d'avoir été à mes côtés tout au long de ce séjour en France. Merci de m'avoir soutenu, spécialement dans ces derniers temps, où la pression du travail s'est accumulée. Merci de m'avoir donné une motivation pour poursuivre, et d'avoir partagé avec moi cette grande aventure…. Quelle aventure !!!

En fin, comme diraient mes filles : « Ç'a été bien ce truc, machin, bidule, chouette ».

Table des matières

Chapitre I

Introduction générale

I.1. Contexte général

Dans le contexte actuel, où l'Internet est en pleine expansion, de nouvelles formes d'interaction entre utilisateurs deviennent possibles, ce qui a donné naissance à de nouveaux sujets de recherche dans le domaine du CSCW (Computer Supported Cooperative Work – travail coopératif assisté par ordinateur). Les outils développés dans ce contexte ont pour but principal de proposer de nouveaux paradigmes de travail en équipe et de développer de nouvelles applications coopératives qui s'appuient sur des réseaux informatiques, dont en particulier l'Internet.

Ainsi, depuis quelques années, des outils tels que des visioconférences, des tableaux blancs partagés, des partages d'application, des éditeurs coopératifs et des environnements intégrés de travail coopératif ont fait leur apparition et ont connu un certain succès, tant au niveau de produits académiques que commerciaux [Joslin-00] [Ishida-98] [Vogelsang-01] [Jaczynski-99].

Nous assistons par ailleurs depuis les années 90s à un essor colossal du World Wide Web, qui outre les services à grande échelle qu'il offre sur le réseau Internet, a grandement influencé la manière dont les applications sont développées aujourd'hui.

Partant de cette constatation, et vu que l'accès au Web peut être aujourd'hui considéré comme omniprésent (*ubiquitous* en anglais), il semble naturel de s'appuyer sur le Web pour développer de nouvelles applications coopératives qui permettraient à plusieurs utilisateurs de coopérer en ligne, et en particulier de partager la connaissances des pages qu'ils sont en train de visiter. C'est dans ce contexte général que se situe notre problématique de recherche.

I.2. Problématique abordée

Malgré son succès, il est reconnu que le Web présente un certain nombre de contraintes, en particulier en terme de facilités de coopération qu'il pourrait offrir à ses utilisateurs. Le Web est ainsi constitué d'un ensemble de serveurs répartis dans le monde entier qui hébergent des documents et qui reçoivent des requêtes de la part d'utilisateurs qui désirent rapatrier et consulter ces documents. Le mode d'utilisation classique du Web s'appuie clairement sur le paradigme client/serveur qui est mis en œuvre grâce au protocole standard HTTP et toute la pile TCP/IP sous-jacente.

Ce mode de fonctionnement n'admet pas d'interactions entre les clients, sauf si ces clients peuvent accéder à un espace de travail partagé qui sera en général centralisé dans un serveur (pensons par exemple à une application de tableau blanc). Les utilisateurs du Web,

quand ils naviguent, sont ainsi isolés les uns des autres. Ils ne disposent d'aucun moyen simple au niveau du Web pour consulter de manière synchronisée des pages Web ou même pour partager la connaissance des pages qu'ils ont visitées. Ainsi, ils ne peuvent en général communiquer entre eux et partager de l'information que par l'intermédiaire d'outils de communication qui sont externes au service Web de base.

Notre objectif, sujet de notre travail de thèse, est ainsi de définir un service, à mettre en œuvre sur le Web, qui permette à un ensemble d'utilisateurs de synchroniser leur navigation et de partager ainsi la connaissance des pages qu'ils visitent.

Une première solution que l'on pourrait envisager pour offrir un tel service consisterait à utiliser un outil de partage d'application, comme l'outil *VNC* (Virtual Network Computing – réseau virtuel de calcul) initialement développé à l'Université de Lancaster (R.U.) [VNC] [Li-00a] [Li-00b] [Richardson-98] qui a ensuite été intégré à la plate-forme collaborative *Platine* [Raymond-04] [Platine] développée au LAAS-CNRS. Le logiciel *VNC* permet de visualiser le fonctionnement d'un ordinateur *A* et d'interagir avec lui à partir de n'importe quel autre ordinateur raccordé à l'Internet. Ainsi, si un utilisateur effectue une navigation HTTP sur l'ordinateur *A*, l'ensemble des utilisateurs raccordés à *A* via l'application *VNC* pourra également voir le résultat de cette navigation. Cette solution, qui semble répondre, au moins en partie, à la problématique posée, présente plusieurs inconvénients sérieux :

- ce n'est pas seulement l'application « navigateur Web » qui s'exécute dans le serveur qui est partagée mais tout le bureau de l'utilisateur utilisant l'ordinateur *A* ;
- ce n'est pas seulement une page Web qui est partagée mais son affichage sur l'écran de l'ordinateur *A* ; ainsi, les différents utilisateurs n'ont pas loisir de visualiser la partie de la page qui les intéresse, car tout l'affichage est fortement synchronisé (ils ne peuvent pas indépendamment jouer avec les ascenseurs présents au niveau de la fenêtre du navigateur) ;
- la synchronisation des pages est (beaucoup trop) forte du point de vue de l'affichage des pages, mais (beaucoup trop) faible vis à vis des relations de synchronisation qui peuvent être établies entre les utilisateurs ; un outil de partage d'applications s'appuie essentiellement sur un paradigme maître – esclave, même si le statut de maître peut passer d'un utilisateur à l'autre – un utilisateur va pouvoir interagir avec le serveur à condition qu'il en obtienne le droit par un mécanisme de contrôle de droit d'accès (floor control) [Dommel-99] [Bullinger-97] [Dommel-97] [Rodriguez-02], mais de toute façon l'application partagée s'exécutera toujours dans le serveur ;
- la mise en œuvre d'un outil de partage d'application nécessite une bande passante importante au niveau du réseau et le serveur qui abrite l'application partagée doit en général être plus performant que les autres postes de travail.

Pour toutes ces raisons, nous avons rejeté la solution du partage d'applications.

Notre objectif est de définir un système de navigation qui permette à des groupes d'utilisateurs de synchroniser leur navigation de la manière la plus flexible possible, sans pour autant avoir à synchroniser la manière dont les pages leurs seront présentées. En d'autres termes, nous souhaitons synchroniser l'accès à des pages Web et non pas la présentation des fenêtres des navigateurs. Nous appelons un tel système, un système de navigation coopérative.

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 système de navigation coopérative complètement original qui permet de définir des relations de synchronisation entre utilisateurs d'une manière à la fois très simple et complètement dynamique ;
2. la formalisation, au moyen d'automates étendus et de réseaux de Petri, du modèle de synchronisation proposé qui est au cœur de notre proposition ; une démarche originale, au moyen de scripts, permet d'engendrer de manière automatique les réseaux de Petri associés à différentes configurations du modèle de synchronisation (selon le nombre d'utilisateurs considérés, selon les primitives de synchronisation considérées, etc) ;
3. La vérification formelle du modèle formel précédent, au moyen des outils TINA et ALDEBARAN, nous permettant de vérifier, dès la conception, le bon fonctionnement des mécanismes de synchronisation proposés. Différentes configurations ont été envisagées pour pallier le problème d'explosion combinatoire auquel nous avons du faire face ;
4. une approche conceptuelle de type « top-down » pour définir l'architecture du système de navigation coopérative en nous appuyant sur le profil UML/SDL supporté par l'outil TAU G2 de Telelogic ;
5. l'implémentation d'un logiciel, appelé CoLab, pour mettre en œuvre le système de navigation coopérative proposé ; à ce jour, CoLab est opérationnel et supporte une grande partie des fonctionnalités décrites dans ce mémoire de thèse ;
6. une analyse critique de l'implémentation réalisée avec plusieurs résultats concernant l'évaluation de performances du serveur Proxy de CoLab ;
7. différentes ouvertures pour faire évoluer notre proposition de système de navigation coopérative concernant d'une part une implémentation répartie de CoLab sur plusieurs serveurs Proxy et d'autre part le déploiement de CoLab sous la forme d'un service Web.

I.4. Plan de thèse

Ainsi notre thèse vise à proposer un système de navigation coopérative permettant à un ensemble d'utilisateurs de consulter de manière synchronisée des pages Web selon des règles de synchronisation qui peuvent être mises à jour de manière dynamique au sein d'une session coopérative.

Le mémoire de thèse est structuré en quatre 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 analysons différentes propositions faites dans la littérature pour proposer des solutions de navigation coopérative et les comparons aux idées qui ont été à l'origine de notre proposition. Dans ce chapitre nous présentons également l'actualité du Web d'un point de vue technique. Nous détaillons en particulier les bases du langage HTML couramment utilisé pour définir des pages Web et celles du protocole HTTP en charge de rapatrier vers un client des pages Web présentes sur un serveur Web distant. Nous utiliserons ces deux technologies dans le déploiement de notre système de navigation coopérative.

Le deuxième chapitre, intitulé *Spécification et vérification formelle du modèle de synchronisation*, introduit le modèle de synchronisation que les utilisateurs enregistrés

dans une session de navigation coopérative vont pouvoir utiliser pour établir des relations de synchronisation entre eux. Différentes requêtes de synchronisation ont ainsi été définies et formalisées. Différents modèles sont présentés dans ce chapitre, un modèle local du rôle de chacune de ces primitives du point de vue d'un utilisateur quelconque de la session coopérative (modèle à base d'automate étendu) et un modèle global (modèle à base de réseaux de Petri). Différentes propriétés de ce modèle sont démontrées au moyen d'automates quotients dérivés des graphes de marquage des réseaux de Petri (automates quotients par rapport à l'équivalence observationnelle selon les événements que l'on souhaite rendre visibles).

Le troisième chapitre, intitulé *Architecture du système de navigation coopérative*, présente l'architecture du logiciel mettant en œuvre notre proposition de navigation coopérative. Partant d'une description intuitive et informelle de cette architecture, nous proposons une démarche pour la formaliser au moyen du 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 validons 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 quatrième chapitre, intitulé *Implémentation et déploiement de CoLab*, présente les différentes techniques logicielles utilisées pour l'implémentation de CoLab. Il rentre ensuite dans les détails de l'implémentation réalisée. Nous détaillons en particulier la façon dont la synchronisation des utilisateurs est gérée, et les mécanismes utilisés pour garantir la cohérence de l'état de synchronisation d'une session. Nous proposons également différents résultats concernant le déploiement de CoLab, comme par exemple : la configuration initiale du serveur Proxy et des clients, l'évaluation des performances de CoLab dans le cas de l'implémentation avec un serveur Proxy unique, le déploiement d'une plate-forme multi-serveurs Proxy et le déploiement du service CoLab sous la forme d'un service Web.

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 de la navigation coopérative sur le Web, ainsi que les techniques classiques utilisées pour naviguer sur le Web.

L'idée principale derrière le concept de la navigation coopérative consiste à permettre à des utilisateurs qui sont en train de naviguer sur le Web d'interagir entre eux, dans le but de synchroniser leur activité de navigation et/ou de partager la connaissance des pages qu'ils visitent. Plusieurs domaines d'application pourraient bénéficier d'un tel service, comme par exemple le E-Learning [Baudin-04] [Baudin-03], domaine dans lequel s'est situé le projet européen Lab@Future [Lab@Future] auquel nous avons contribué.

Dans un premier temps, nous présenterons une analyse de différentes propositions que nous avons trouvées dans la littérature, et qui, d'une manière ou d'une autre, offrent des solutions pour élargir les possibilités actuelles de navigation sur le Web.

Dans un deuxième temps, nous présenterons de manière détaillée les deux techniques standards qui sont à la base du déploiement du Web : le langage HTML pour programmer des pages Web et le protocole HTTP pour permettre à un client de rapatrier des ressources Web d'un serveur distant.

II.2. Applications coopératives sur le Web

Nous avons effectué une recherche bibliographique dans le but de prendre connaissance de différentes approches, qui proposent d'une manière ou d'une autre, d'étendre les capacités du Web pour qu'il puisse être utilisé en tant qu'outil coopératif. Dans le cadre de cette recherche nous avons identifié plusieurs propositions qui abordent ce problème en fonction de différents points de vue. Nous les avons classées en deux grandes catégories : (i) des systèmes offrant des solutions coopératives spécifiques et (ii) des systèmes offrant des environnements de travail intégrés, et qui permettent à des utilisateurs de synchroniser leur navigation sur le Web. D'autres produits académiques existent également que nous ne développerons pas ici comme Platine [Platine], Isabel [Isabel], VRVS [VRVS].

II.2.1. Solutions spécifiques de navigation coopérative

Parmi les propositions que nous avons analysées nous avons trouvé entre autres des mécanismes de synchronisation de la navigation, des mécanismes d'annotation des pages Web visitées et des mécanismes de création de voisinages virtuels. En fait plusieurs des

caractéristiques de ces propositions nous ont servi de source d'inspiration dans la conception et la définition de certains aspects de la proposition que nous développons dans ce mémoire de thèse.

Nous présentons dans ce paragraphe, quelques uns des projets qui on été les plus significatifs en fonction de nos objectifs de recherche, expliquons leurs caractéristiques et fonctionnalités principales et faisons quelques commentaires pour chacun d'entre eux.

II.2.1.1. CoBrow

Le projet CoBrow [Sidler-97] a pour but de créer de nouvelles formes de coopération sur l'Internet. Les auteurs proposent d'élargir le modèle actuel de fonctionnement du Web en ajoutant le concept de « lieux de rencontre » (meeting places), où des utilisateurs peuvent se rencontrer, partager de l'information et coopérer à des projets communs. L'approche choisie consiste à associer les utilisateurs à un emplacement sur le Web. Cette association s'appuie sur le document en cours de visualisation par un utilisateur dans son navigateur. Différents attributs, tels que l'intérêt, la langue, le temps de présence entre autres peuvent être associés aux utilisateurs. À partir de ces emplacements et de leurs attributs, CoBrow crée des voisinages virtuels, dans lesquels les utilisateurs peuvent avoir conscience de la présence des uns et des autres, et où des relations de collaboration peuvent être crées au moyen de différents outils disponibles.

L'idée générale proposée par ce projet est très intéressante, puisque la formation des voisinages virtuels semble être une bonne approche pour faciliter le « hasard heureux » (*serendipity* en anglais) et permettre à chaque utilisateur d'obtenir des informations pertinentes en tenant compte des intérêts similaires d'autres utilisateurs.

Nous pensons que la création de voisinages virtuels devrait tenir compte d'autres données que le simple URL visité ou le profil des utilisateurs. Pour qu'un tel voisinage soit plus effectif il faudrait tenir compte de l'information contenue dans les pages visitées, de manière à créer des groupes de travail potentiels à partir de sujets similaires. Évidemment l'implantation d'un tel mécanisme serait beaucoup plus complexe puisqu'il faudrait faire une analyse sémantique du contenu des documents, au moyen par exemple de techniques de l'intelligence artificielle.

À partir du concept de voisinage, il serait également désirable de disposer de facilités pour que des utilisateurs, appartenant au même voisinage, puissent synchroniser leur activité de navigation.

II.2.1.2. E-CoBrowse

E-CoBrowse [Chong-00] est un cadre de travail extensible de co-navigation. Les auteurs proposent le développement d'un système permettant d'ajouter des extensions multi utilisateurs aux navigateurs Web, principalement MicroSoft® Internet Explorer® et Netscape Navigator®. Ceci comprend des « chatpointers », qui permettent aux utilisateurs d'annoter simultanément la page Web qu'ils visitent.

Trois modes de fonctionnement sont proposés dans cette approche : « leader », « follower » et « standalone ». Le premier mode correspond à l'implémentation d'un maître, le second à l'implémentation d'un esclave, et le troisième à un utilisateur qui n'influence pas le système et qui ne subit aucune influence du système (aucune synchronisation n'est réalisée).

La proposition est par elle-même très intéressante. Cependant, un détail attire notre attention, à savoir qu'il est strictement nécessaire, pour implémenter les extensions proposées, d'utiliser des pages codés en DHTML, ce qui réduit la généralité de la proposition, puisque restent exclues les pages codées en HTML de base et celles codées

avec d'autres variantes de ce langage. En fait la proposition va au-delà puisque l'implémentation d'E-CoBrowse nécessite un codage spécifique au navigateur utilisé, ce qui réduit encore la flexibilité de la proposition.

Un détail qui reste encore un peu obscur à la lecture de la documentation disponible est de savoir si la synchronisation de la navigation Web est effective, c'est à dire si suite à une navigation d'un utilisateur « leader », la même ressource est chargée et affichée immédiatement dans les navigateurs des utilisateurs « follower », ou si ces ressources ne sont chargées que périodiquement (la page des « followers » est actualisée sur la base d'une période de temps), comme le laisse entrevoir la documentation.

II.2.1.3. Let's Browse

Les auteurs de ce projet [Lieberman-99] partent de l'hypothèse qu'il existe probablement souvent de nombreux utilisateurs dans le monde qui recherchent des informations similaires. À partir de cette hypothèse, ils proposent qu'à un instant donné le système puisse aider à faciliter des rencontres entre personnes partageant des intérêts de navigation similaires. Ils proposent donc la création d'un agent pour assister un groupe d'utilisateurs qui naviguent. Le but de l'agent est de proposer des pages Web aux utilisateurs en fonction de la navigation des utilisateurs ayant des domaines d'intérêt commun.

Ce projet est construit comme une extension de l'agent de navigation Web appelé Letizia. Cet agent effectue une recherche en largeur (« breadht first ») en temps réel de ressources associées à la page courante de l'utilisateur, en filtrant des pages candidates selon un profil appris de l'observation des actions de navigation de l'utilisateur.

Ce projet ressemble au projet CoBrow introduit auparavant et présente une problématique similaire en termes d'implémentation.

II.2.1.4. PROOF

Ce modèle [Cabri-99] a été proposé par Giacomo Cabri et al. à l'Université de Modena, en Italie. Il consiste en un système qui permet la synchronisation de la navigation sur le Web. Son architecture s'appuie sur l'existence d'un serveur Proxy, qui traite les requêtes des utilisateurs. Du côté utilisateur, nous avons un applet qui permet au navigateur d'un client de rester en communication avec le serveur Proxy, et d'échanger avec lui des messages de contrôle. La synchronisation de la navigation Web est réalisée par modification des ressources rapatriées avant de les envoyer au client.

Deux modes de navigation coopérative sont permis : faiblement couplé et fortement couplé. Dans le mode faiblement couplé, chaque utilisateur navigue de sa propre initiative, mais il peut consulter à tout moment les URLs récemment visités par les autres utilisateurs connectés à la même session. Le mode fortement couplé s'appuie sur un schéma maître esclave, de telle façon que dans les navigateurs de tous les esclaves sont présentées les mêmes pages Web que celles visitées par le maître. Toute navigation indépendante de la part des esclaves reste interdite tout au long d'une session.

Les utilisateurs peuvent enrichir leurs interactions et communiquer entre eux par l'utilisation de deux outils intégrés à la plate-forme : un système de messagerie instantanée (chat) et un tableau blanc. D'autres outils peuvent être intégrés à la plate-forme de base, qui est programmée en Java.

Cette proposition nous a semblé dès le début de nos travaux très intéressante et a été une de nos sources d'inspiration principales. Deux points nous paraissent cependant un peu contraignants concernant le fonctionnement de cette plate-forme. Le premier est que le mécanisme de synchronisation proposé est très rigide, puisque, une fois qu'un maître a été

défini, il reste maître tout au long de la durée de la session. Le second est que l'architecture proposée est très centralisée, ce qui peut poser des problèmes de passage à l'échelle.

À partir de ce modèle de navigation coopérative sur le Web, M. Cabri et son équipe ont proposé toute une série d'expériences s'appuyant sur cette technologie, démontrant ainsi tout son intérêt.

II.2.1.5. WABX

Ce projet [WABX] a été proposé par l'EISTI (École Internationale des Sciences du Traitement de l'Information) à Cergy-Pontoise (France). Il consiste en la création d'un ensemble d'outils pour la coopération à distance en ligne, dont un outil de navigation coopérative sur le Web au cœur de cette proposition. Apparemment faute de financement, ce projet a été arrêté il y a environ 2 ans. Peu d'informations restent disponibles aujourd'hui et plusieurs liens disponibles auparavant sur le Web ne le sont plus.

Au niveau de la navigation coopérative, les concepteurs de WABX proposent un outil qui permet la synchronisation de la navigation Web à un niveau très fin, incluant, par exemple, la synchronisation de la barre de défilement du navigateur. Dans ce but, l'implémentation s'appuie sur l'utilisation d'un Plug-in et d'applets signés, ce qui, à priori, pose un certain nombre de problèmes.

En ce qui concerne l'utilisation d'un Plug-in, le principal inconvénient est que son implantation dépend de l'outil logiciel de navigation que l'on l'utilise. Ainsi, par exemple, il devrait exister un Plug-in différent pour chaque type de navigateur présent sur le marché, et pour chaque version de chacun d'eux, ce qui rend difficile une utilisation à grande échelle d'une telle plate-forme.

Par ailleurs, l'utilisation d'applets signés peut représenter des risques de sécurité au niveau des utilisateurs, et pour cette raison tous les utilisateurs ne sont pas prêts à autoriser leur exécution dans leurs propres navigateurs. Habituellement les applets sont exécutés dans un environnement sûr d'exécution puisque aucun accès aux ressources de la machine hôte ne leur est accessible. Il existe toutefois la possibilité d'autoriser un applet à avoir un accès illimité aux ressources de la machine, et qui consiste à la signer. Le problème est qu'un applet signé dispose de tous les droits, ce qui représente un risque potentiel pour les utilisateurs.

Nous pensons également que le niveau de synchronisation proposé par WABX n'est pas réellement adapté à un système de navigation coopérative. Il nous paraît ainsi plus intéressant de synchroniser l'accès à des pages que de synchroniser la présentation de chacune de ces pages par un navigateur, ce dernier type de fonctionnalité pouvant être proposé, avec certaines contraintes il est vrai, par un système de partage d'applications.

II.2.1.6. WebSplitter

Ce modèle [Han-00] offre un cadre de travail XML unifié qui offre des moyens pour faire de la navigation Web multi-dispositifs et multi-utilisateurs. La principale contribution de cette proposition est la capacité de créer, à la volée, des vues partielles et personnalisées de la même page Web en fonction des droits d'accès des utilisateurs et des capacités techniques de l'équipement dont les utilisateurs disposent.

Du côté client, il existe une un applet dans le navigateur de chaque utilisateur, ce qui permet à ce dernier de garder une ligne de communication avec le serveur et d'échanger avec lui des messages de contrôle.

Chaque page Web est accompagnée d'un fichier de politiques (« policy file »), qui contient des règles permettant de mettre en correspondance des balises d'hypertexte à des groupes de privilèges. Un seul fichier de politiques peut définir plusieurs groupes de

privilèges pour une seule page Web. Un groupe de privilège peut être conceptualisé comme étant un rôle, de telle façon, que lorsqu'un utilisateur s'identifie comme appartenant à un certain groupe de privilèges, il acquiert les droits correspondants.

Un service de découverte a été implémenté au niveau « middleware », permettant à WebSplitter de découvrir les dispositifs que les utilisateurs ont à leur disposition, de manière à adapter les présentations des pages Web. Ainsi, par exemple, un utilisateur ayant accès à une page Web par l'intermédiaire d'un PDA recevra une version allégée de cette ressource.

Cette proposition présente selon nous deux intérêts. Le premier intérêt est le concept de groupe de privilèges, qui permet de définir des droits d'accès et des besoins d'adaptabilité, au niveau de groupes d'utilisateurs et non pas d'utilisateurs individuels. Ce concept fait partie des définitions de RBAC (Role-Based Access Control – contrôle d'accès s'appuyant sur des rôles) [Wang-99].

Le second intérêt est l'adaptation des présentations, ce qui offre une grande flexibilité garantissant que jamais un utilisateur ne recevra une page Web que son dispositif d'accès ne pourra présenter. Cette adaptabilité devrait également tenir compte de la bande passante de la liaison d'accès de l'utilisateur (liaison Ethernet haut débit, liaison WiFi, liaison GPRS, etc.).

Un inconvénient de cette proposition est que les politiques sont définies au niveau d'un fichier associée au fichier principal de la ressource, ce qui introduit un certain degré de complexité au niveau de la spécification des politiques d'accès et d'adaptabilité.

II.2.1.7. Wiki Wiki Web

Ce projet [WikiWikiWeb] met en place un système coopératif pour créer des pages sur le Web. Cela signifie que n'importe quel utilisateur enregistré peut, depuis n'importe quel endroit du monde, et à n'importe quel moment, ajouter des nouvelles pages Web au système, et il peut également, s'il dispose des privilèges nécessaires, modifier et éliminer des pages existantes. Ce projet ouvre des possibilités très intéressantes puisqu'il permet de créer de manière coopérative des bases d'information qui sont mis à jour régulièrement et qui sont consultables sur le Web. Ce projet a donné naissance à la WikiPédia [WikiPedia] qui est une encyclopédie créée dynamiquement par ses propres utilisateurs.

II.2.2. Environnements Web intégrés de travail coopératif

Nous présentons maintenant des systèmes proposant des solutions intégrées pour gérer la coopération entre plusieurs utilisateurs raccordés sur le Web. Les deux caractéristiques communes de ces systèmes sont : (i) ils sont constitués de plusieurs composants de communication permettant aux utilisateurs d'interagir entre eux, (ii) ils sont accessibles depuis un simple navigateur Web.

Parmi les systèmes les plus représentatifs que nous avons identifiés dans cette catégorie nous pouvons citer *NetDive*, *Microsoft Office Live Meeting* (autrefois connu sous le nom de *PlaceWare*) et *WebEx*, qui sont tous des produits commerciaux.

II.2.2.1. NetDIVE

Ce système [NetDive] est un environnement intégré de collaboration, destiné à des activités commerciales, qui est composé de quatre applications principales :

- *WeMeeting :* une messagerie instantanée permettant à des utilisateurs de rester en contact les uns avec les autres, tout en gardant conscience de leur présence (notion exprimée par le terme awareness en anglais).

- *eAuditorium :* un outil permettant d'établir des facilités de téléconférence via le Web avec jusqu'à 3000 participants. Il permet l'organisation de réunions s'appuyant sur le Web, les utilisateurs étant catégorisés en modérateurs, audience et orateurs en fonction de leur nom et mot de passe. Les échanges entre participants peuvent être réalisés en VoIP, par des messages textuels, par un outil de partage d'applications (navigation Web synchronisée ou présentation synchronisée de transparents PowerPoint) et un outil de tableau blanc partagé. Il est également possible de mettre en place des votes et des enchères.
- *CallSite :* un système s'appuyant sur le Web capable de permettre à des utilisateurs de communiquer entre eux tout en visitant un site Web. Le principe de fonctionnement est simple : un utilisateur ayant besoin de contacter quelqu'un de l'entreprise dont la page Web est actuellement affichée dans son navigateur, peut cliquer sur un bouton particulier, action par laquelle il entre en contact direct avec la personne désirée.
- *SiteSticky :* un serveur de chat capable de gérer des communications vocales, des discussions immersives par l'utilisation d'avatars, etc.

Toutes ces applications ont certains éléments en commun, et d'autres sont spécifiques à la solution du problème posé. Dans le but de donner un aperçu général des capacités techniques de ces applications, nous présentons, dans le tableau de la *Figure 1,* une synthèse des principales caractéristiques de chacun de ces éléments.

Module / **Caractéristique**	**WeMeeting**	**eAuditorium**	**CallSite**	**SiteSticky**
Audioconférence via VoIP multi utilisateurs.	●			
Boutons d'appel instantané.			●	
Capacité de modération dans une conférence vocale.		●		
Capacités de filtrage de messages.				●
Communication textuelle instantanée.	●		●	●
Communication sécurisée.	●	●	●	●
Communication vocale.	●	●	●	●
Conférence textuelle avec capacité de modération.		●		
Gestion des utilisateurs via l'accès à une base de donnés SQL.	●			●
LiveTrax (visualisation en ligne des utilisateurs visitant actuellement le site).			●	
Messagerie hors ligne.	●			
Modérateur, membres de l'audience, orateurs et screeners (qui assistent le modérateur pour de très grandes conférences).		●		
Conscience de la présence des utilisateurs à l'échelle globale.	●			
Outil de génération de rapports.			●	●
Partage d'applications.	●	●	●	

Partage de documents via un tableau blanc partagé.	●	●	●	●
Possibilité d'activer/désactiver l'utilisation d'un mot de passe.				●
Possibilité d'un mode Mini-Admin pour un contrôle par salon.				●
Possibilité d'utilisation des styles de police.	●			●
Possibilité de créer des salons protégés par mot de passe.				●
Présentation à distance d'URLs (push).	●		●	●
Système d'administration s'appuyant sur le web.			●	●
Chat immersif par l'utilisation d'avatars 2D.				●
Chat privé (1-1).				●
Technologie légère du côté des utilisateurs par utilisation d'applets.		●		●
Transfert d'appels.	●		●	
Transfert instantané et partage de fichiers.	●		●	
Utilisable à travers des pare-feu.		●	●	●
Visualisation en ligne du nombre d'utilisateurs connectés.				●
Votes et enchères en direct.		●		

Figure 1. Tableau comparatif des caractéristiques des outils de NetDive

En ce qui concerne notre sujet d'intérêt, la navigation coopérative, nous pouvons constater qu'il est possible avec la plate-forme NetDive de forcer l'affichage d'une page Web quelconque dans l'interface des utilisateurs suivant actuellement une présentation. Notons que cet affichage ne se fait pas dans le navigateur Web, mais dans l'environnement de travail défini par la plate-forme.

II.2.2.2. Microsoft Office Live Meeting (autrefois connu comme PlaceWare)

Le système, qui s'appelait à l'origine PlaceWare, a été récemment racheté par Microsoft® et intégré dans l'environnement de travail Office®, sous le nom de Microsoft Office Live Meeting® [MOLM] (il fait partie intégrante de la version 2003 d'Office).

Il s'agit d'un environnement collaboratif dans lequel il est possible de partager des documents natifs Office, et qui comprend un certain nombre d'outils de communication, tel qu'un tableau blanc partagé et un chat.

Quelques unes des caractéristiques principales de ce système sont présentées ci-dessous :

- *Visionneuse PowerPoint :* elle permet à plusieurs utilisateurs de visualiser de manière synchronisée des présentations PowerPoint, y compris les animations qui peuvent être contenues dans ces présentations. L'affichage peut se réaliser en plein écran ou dans une fenêtre dédiée.
- *Visionneuse de documents :* elle permet à plusieurs utilisateurs de visualiser de manière synchronisée tout type de document imprimable.

- *Partage de bureau :* il permet de contrôler l'exécution d'applications à distance. Le contrôle du bureau partagé peut être passé dynamiquement d'un utilisateur à l'autre.
- *Outils interactifs :* des outils tel qu'un gestionnaire de questions permettant à des utilisateurs de visualiser et de répondre à des questions de façon individuelle ou collective. D'autres outils interactifs permettent de gérer des votes en temps réel, des indicateurs d'humeur, des chats, des annotations, des tableaux blancs partagés, etc. ;
- *Intégration avec Outlook et Messenger :* Microsoft Office Live Meeting permet la planification de réunions à partir de différents outils Microsoft dont Outlook®, Word®, Excel®, PowerPoint®, Project®, et Visio®, ou par l'intermédiaire d'outils de communication instantanée, tels que Messenger® et MSN Messenger® ;
- *Intégration avec Lotus Notes :* il est possible de planifier des réunions par utilisation de Lotus Notes, en vérifiant la disponibilité des participants et en réalisant un suivi des traces de leurs confirmations ;
- *Diffusion audio sur Internet :* c'est une alternative à l'audio conférence qui permet aux utilisateurs d'envoyer des flux audio sur l'Internet ;
- *APIs de Live Meeting :* cette API permet d'intégrer Live Meeting à d'autres applications ;
- *Meeting Lobby :* c'est un lieu ne nécessitant pas de réservation où il est possible de rencontrer d'autres utilisateurs ;
- *Live Meeting Replay :* c'est une facilité qui permet d'enregistrer une présentation faite au moyen de Live Meeting pour pouvoir la reproduire ultérieurement ;
- Politiques de sécurité: le chiffrement des mots de passe est une caractéristique primordiale dans ce type de système. Il est également possible de définir des dates d'expiration pour certaines des ressources disponibles.

Nous avons utilisé et testé la version de démonstration disponible sur la page Web de Live Meeting. Il s'agit d'un applet qui contient un environnement intégré de collaboration avec les outils cités précédemment. Un aperçu de l'interface de ce système est présenté dans la *Figure 2*.

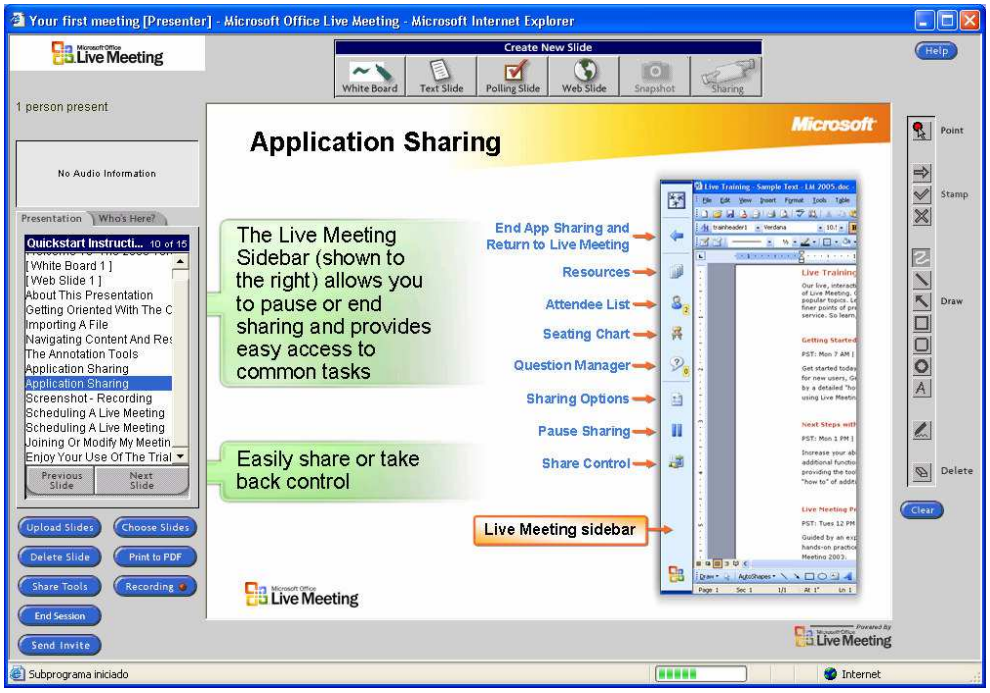


Figure 2. Aperçu de l'interface de Microsoft Office Live Meeting

II.2.2.3. WebEx

WebEx [WebEx] est un outil pour la réalisation de réunions virtuelles sur le Web, qui offre les fonctionnalités suivantes :

- le partage de présentations effectuées dans n'importe quel format ;
- l'échange de fichiers et de documents ;
- le partage de bureau et la prise de contrôle à distance des applications s'exécutant sur un serveur de partage d'applications ;
- des possibilités de communication audio.

L'architecture de WebEx est construite autour d'un composant principal, auquel il est possible d'ajouter de nouvelles fonctionnalités sous la forme d'extensions. Les composants à la base de WebEx sont ainsi :

- *Meeting Center :* c'est le composant central de l'architecture de WebEx qui offre, à partir d'un navigateur Web, des facilités de communication audio et vidéo pour de petits groupes de travail ;
- *Event Center :* c'est le module qui permet l'organisation de présentations pour de plus grandes audiences (valeur cible : 3000 participants) ; il comprend la planification, la promotion, la diffusion de ces présentations et tout ce qui concerne le traitement post-présentation. Parmi les facilités proposées, nous trouvons le partage d'applications, la diffusion de présentations à base de transparents en format PowerPoint®, l'intégration d'outils de téléphonie classique, des mécanismes de votes, d'enregistrement et de reproduction de présentations, etc. ;
- *Training Center :* c'est le composant de WebEx permettant la diffusion « live » ou sur demande de formations. Il dispose des outils nécessaires pour générer une ambiance pédagogique propice à un environnement d'apprentissage, tels que des outils pour des interactions de type question-réponse, intégration avec Outlook® et PowerPoint®, outils de votes, etc. ;
- *Support Center et SMARTtech :* c'est un composant qui permet à un représentant du service technique de voir, diagnostiquer et résoudre des problèmes « on-line ». Les fichiers de l'utilisateur à l'origine de la requête de service peuvent être transmis pour une analyse post-mortem ; avec l'autorisation du client, le personnel de service technique peut exécuter des logiciels de diagnostic et installer les corrections nécessaires ;
- *Sales Center :* c'est le composant de WebEx orientée à la vente directe. Il offre des outils de communication de base, des outils spécialisés, tel qu'un indicateur d'attention du client, et la possibilité d'inviter en ligne des spécialistes pour faire des démonstrations spécifiques.

Un aperçu de l'interface de WebEx est présenté dans la *Figure 3*.

D'autres sources d'information ont été consultées lors de notre étude, telles que [Wang-99], [Steinfield-99], [Bouthors-99], [Takahashi-00], [Sanranch-00], [Sakamoto-00], [Boticario-00], [Tolksdorf-01], [Reif-01] et [Cabri-01].

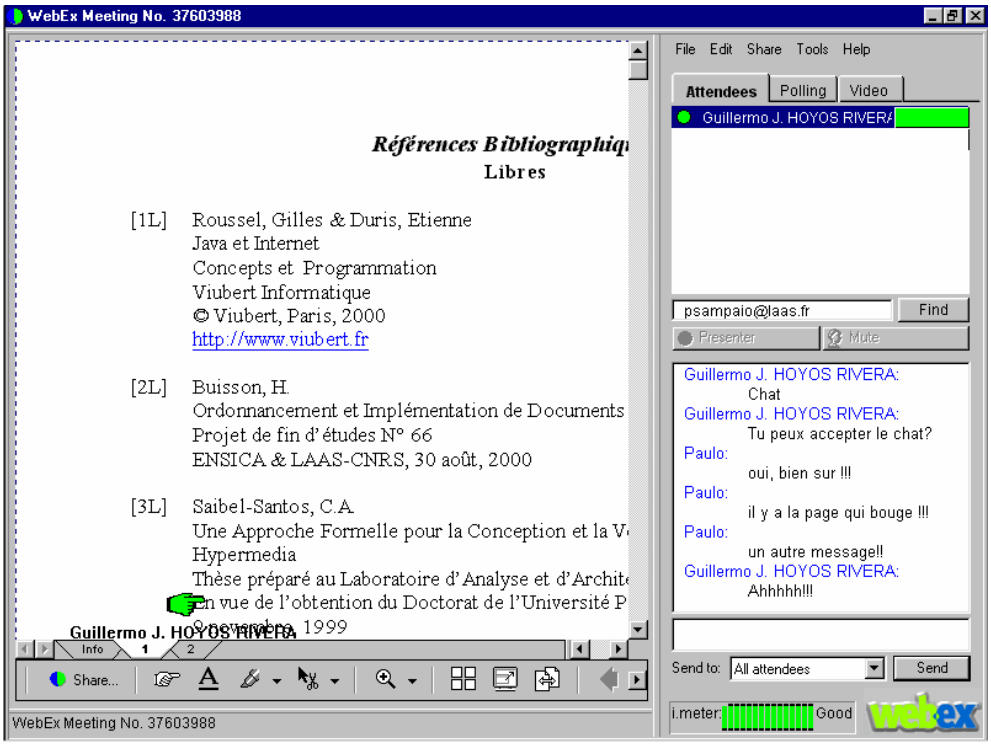


Figure 3. Aperçu de l'interface de WebEx

II.2.3. Positionnement de notre proposition

Parmi les différentes solutions que nous venons d'analyser nous avons identifié différentes approches permettant, d'une manière ou d'une autre, de gérer des interactions entre utilisateurs navigant sur le Web. Le concept de synchronisation de la navigation Web de plusieurs utilisateurs nous semble cependant avoir été traité d'une manière plutôt limitée et trop rigide, s'appuyant en général uniquement sur un schéma maître-esclave.

Dans les chapitres suivants où nous allons développer notre approche selon différents aspects comprenant la formalisation du modèle de synchronisation, la définition de l'architecture, l'implémentation et l'évaluation de l'outil réalise CoLab, nous proposons une approche permettant de créer et de supprimer de manière complètement dynamique et répartie des relations de synchronisation entre utilisateurs appartenant à une même session de navigation coopérative.

Avant d'aborder ces chapitres, nous allons rappeler dans le paragraphe suivant le fonctionnement d'un Proxy Web, ainsi que les briques de base du Web, à savoir le langage HTML et le protocole HTTP.

II.3. HTML et HTTP : les briques de construction du Web

Le Web, également connu sous l'acronyme WWW (World Wide Web – toile d'araignée mondiale), est un moyen de communication fonctionnant selon le paradigme client-serveur. D'un côté, nous disposons de machines serveurs, qui exécutent un logiciel serveur capable d'accepter des requêtes et, éventuellement, de les satisfaire grâce à un ensemble de ressources, principalement des pages Web, disponibles localement au niveau de ces serveurs. D'un autre côté, nous disposons de clients, dans lesquels a été installé un logiciel (appelé en général un navigateur) capable d'envoyer des requêtes vers un serveur, et de présenter sur un écran l'information obtenue en réponse à une requête. Cette idée, aujourd'hui tout à fait classique, est illustrée dans la *Figure 4*.

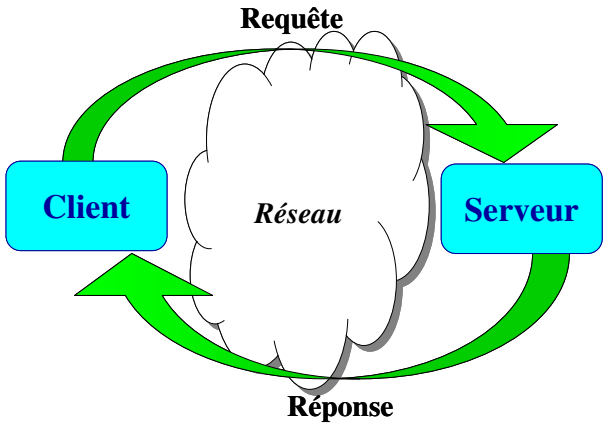


Figure 4. Notion de Client Serveur dans le Web

Les requêtes peuvent éventuellement passer par un intermédiaire, appelé un serveur Proxy [Luotonen-98]. Ce serveur a la capacité de recevoir des requêtes émises par un client, et d'aller chercher, s'il est nécessaire, la ressource demandée sur le serveur d'origine, pour ensuite rediriger la réponse obtenue vers le client, cette opération restant complètement transparente au client. Il suffit de configurer localement le client pour que toute nouvelle requête soit dirigée vers le serveur Proxy, et tout se passe alors comme si la connexion était directement établie avec le serveur d'origine. Ceci est illustré dans la *Figure 5*.

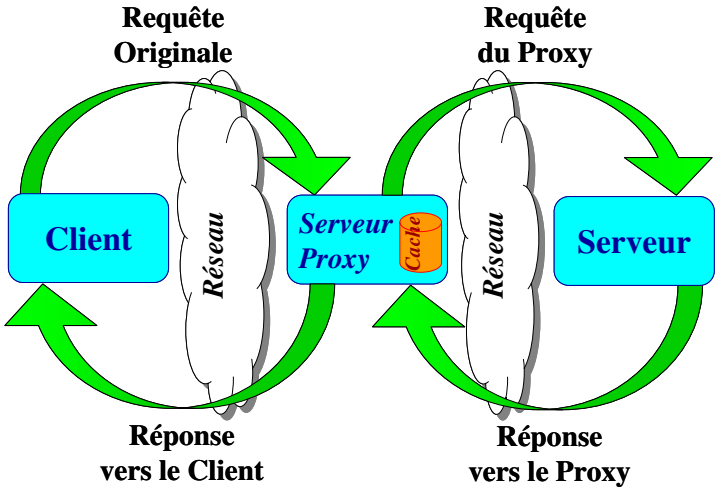


Figure 5. Récupération des ressources à travers un serveur Proxy

Plusieurs raisons peuvent motiver l'utilisation de serveurs Proxy. La principale est de les considérer comme des systèmes de cache offrant la possibilité de stocker localement les ressources rapatriées dans le but de minimiser les besoins de retransmission de ressources à partir du serveur d'origine à travers un réseau, dont la bande passante peut être assez limitée. Dans un tel cas, lors de l'arrivée d'une requête émise par un client, le serveur Proxy vérifie si la ressource demandée est stockée dans un cache local (généralement disque), et si cette copie locale n'est pas encore périmée par rapport à la ressource originale. Si ces deux conditions sont remplies, la requête pourra être satisfaite directement à partir de la copie locale, éliminant ainsi tout besoin de retransmettre la ressource à partir du serveur d'origine. Dans le cas contraire, la ressource est demandée au serveur d'origine, la copie locale est remplacée par une nouvelle version, et la requête du client peut alors être satisfaite.

Dans cette section nous allons discuter en détail des aspects relatifs au langage HTML et au protocole HTTP. Ces deux standards ont été particulièrement importants dans le cadre de notre travail puisqu'ils constituent deux éléments de base sur lequel s'appuie tout système de navigation sur le Web.

II.3.1. Le langage HTML

HTML [HTML] signifie « Hyper-Text Markup Language » (Langage de Marquage d'HyperTexte). Il s'agit d'un langage hiérarchisé de programmation qui s'appuie sur le standard SGML (Standard Generalized Markup Language) [SGML]. SGML correspond à la norme ISO 8879. Ce langage est, à l'heure actuelle, le plus utilisé pour la description de pages Web. Développé à l'origine par Tim Berners-Lee quand il travaillait au CERN [CERN] à Genève, il s'est popularisé grâce au navigateur Mosaic, développé par le NCSA (National Center for Supercomputing Applications) [NSCA], aux Etats-Unis. La version actuelle du standard HTML est la version 4.01 [HTML401].

Le but principal de HTML est de donner aux utilisateurs la possibilité de créer des documents électroniques contenant des textes et des images, certains d'entre eux pouvant être sensibles et contenant ainsi des références vers d'autres documents. Il est possible alors d'accéder à ces derniers par un simple click de souris sur la définition d'un hyperlien. Chacun de ces documents peut se trouver dans n'importe quel serveur Web dans le monde. Celle-ci est l'idée de base de l'hypertexte. La grande popularité de HTML est principalement due à sa simplicité de spécification et d'opération.

Le contenu de fichiers HTML est placé dans des *éléments* `<HTML>`. Ces éléments sont marqués par ce qu'on appelle des balises (tags). Presque tous les éléments HTML comprennent une balise d'ouverture et une balise de fermeture. Le texte situé entre elles est le "domaine de validité" des balises concernées. Les balises sont marquées par des crochets '<' et '>'. La règle d'utilisation de ces balises est comme suit :

- **les balises d'ouverture** : elles se caractérisent par un mot clé, par exemple `<HTML>` ;
- **les balises de fermeture** : elles se caractérisent par un mot clé (le même que dans la balise d'ouverture correspondante) précédé par le caractère '/', par exemple `</HTML>` ;
- **les balises autonomes :** ce sont des balises sans aucun contenu, donc il n'y a pas besoin d'une balise d'ouverture et d'une balise de fermeture, par exemple `<BR>`.

Les éléments peuvent être imbriqués les uns dans les autres. De cette manière se crée une structure hiérarchique. Les fichiers HTML complexes comprennent de nombreuses imbrications. Nous l'illustrons ces idées par la présentation le style d'imbrication des programmes HTML dans la *Figure 6*.

```
<HTML>
  <HEAD>
    <TITLE>
       Titre du programme
    </TITLE>
  </HEAD>

  <BODY>
     Voici le corps du programme…
  </BODY>
</HTML>
```

Figure 6. Exemple de programme HTML

Les éléments HTML peuvent être vus comme des conteneurs, et les domaines de validité qu'ils contiennent ne peuvent pas se recouvrir. Ainsi nous présentons un exemple d'un programme HTML erroné dans la *Figure 7*.

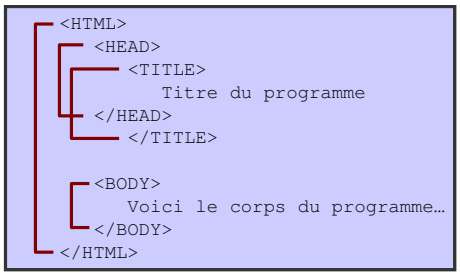


Figure 7. Exemple d'un programme HTML erroné

Les balises d'ouverture et les balises autonomes peuvent contenir des données additionnelles appelées attributs. Ces attributs modifient les propriétés des balises qui les contiennent. Les attributs suivants peuvent affecter les éléments HTML :

- attributs avec affectation de valeur, pour lesquelles existent certaines valeurs permises, par exemple pour `<h1 align="center">` (ici ne sont acceptées que les valeurs left, center, right et justify) ;
- attributs avec affectation de valeur libre, où cependant un type de données ou une certaine convention sont attendus, par exemple `<style type="text/css">` (ici c'est ce qu'on appelle un Type Mime qui est attendu comme valeur). Ou bien <table border="1"> (ici, c'est une mention de valeur numérique qui est attendue) ;
- attributs avec affectation de valeur libre, sans convention particulière, par exemple <p title="Assertion sous réserve"> (ici c'est tout un texte qui peut être affecté) ;
- attributs utilisés seuls, par exemple `<hr noshade>`.

Toutes les valeurs affectées à des attributs doivent figurer entre guillemets. La plupart des navigateurs ne se formaliseront certes pas si les guillemets manquent et le consortium W3C a déjà exprimé des avis très divergents à ce sujet. Pourtant depuis le standard HTML 4.0 les guillemets sont sans conteste prescrits, et celui qui veut écrire du HTML correct doit s'y tenir.

À côté des attributs que l'on ne rencontre qu'avec certains éléments HTML, il y a aussi les attributs qui sont permis avec beaucoup voire presque tous les éléments HTML, et que l'on appelle les attributs universels, comme par exemple `<p id="Introduction">texte</p>`. Par l'utilisation de l'attribut « id= » il est possible de donner un nom à des éléments HTML distincts.

Un fichier HTML traditionnel comprend essentiellement les parties suivantes :

- mention du type de document (mention de la version HTML utilisée) ;
- header (entête par exemple mentions du titre ou similaires) ;
- body (corps - contenu à afficher donc texte avec des titres, des liens, des références de graphique etc.)

Ainsi dans la *Figure 8* nous présentons comme exemple un programme HTML simple.

```
<HTML>
        <HEAD>
                <TITLE>Texte du titre</TITLE>
        </HEAD>
        <BODY>
                Corps du programme….
        </BODY>
</HTML>
```

Figure 8. Un programme simple en HTML

II.3.1.1. Les URL

Un concept très important dans le contexte de la navigation Web est celui d'URL (Uniform Resource Locator – localisateur uniforme des ressources). Les URL sont un sous-ensemble du projet initial de Tim Berners-Lee, les URI (Uniform Resource Identifier – identificateur uniforme de ressources). Une URI a pour objectif de permettre d'identifier une ressource de manière permanente, même si la ressource est déplacée ou supprimée. Jusqu'à présent, les URL sont les seuls URI ayant trouvé une application pratique. Les URIs sont décrits en détail dans la RFC 2396 de l'IETF [RFC2396].

Un URL est utilisé là où l'on veut indiquer l'endroit où se trouve une ressource quelconque, comme par exemple, une image, une page Web, un son, etc. Un URL désigne ainsi le nom d'une ressource sur Internet. Il s'agit d'une chaîne de caractères ASCII imprimables qui se décompose en cinq parties :

- **Le nom du protocole** utilisé pour rapatrier la ressource. Le protocole le plus largement utilisé est HTTP, qui permet d'échanger des pages Web au format HTML. D'autres protocoles peuvent également être utilisés comme FTP, NEWS, MAILTO, GOPHER, etc.
- **Un identifiant et un mot de passe** pour permettre de spécifier les paramètres d'accès à un serveur sécurisé. Cette option est déconseillée car le mot de passe sera directement visible au niveau de l'URL pendant la transmission.
- **Le nom du serveur** pour identifier l'ordinateur hébergeant la ressource demandée. Notez qu'il est possible d'utiliser aussi bien le nom DNS que l'adresse IP du serveur.
- **Le numéro de port** TCP utilisé pour la connexion. Le port associé par défaut à l'utilisation du protocole HTTP est le numéro 80. Si un serveur met à disposition des ressources sur le port 80, le numéro de port devient facultatif dans l'URL.
- **Le chemin d'accès à la ressource** permet au serveur de connaître l'emplacement précis où la ressource est située à l'intérieure de l'arborescence du serveur, c'est-à-dire de manière générale l'emplacement (répertoire) et éventuellement le nom du fichier demandé.

À partir de l'explication précédente une URL a donc la structure présentée dans la *Figure 9*.

Protocole	*Mot de passe*	*Nom du serveur*	*Port*	*Chemin*
http://	user:password@	www.qqch.fr	80	/exemple/index.html

Figure 9. Spécification de la structure d'un URL

Optionnellement un URL peut être suivi d'une liste de paramètres qui servent à envoyer de l'information à une application sur le serveur (un script CGI, par exemple). Dans un tel cas ces données sont précédées par un point d'interrogation ('?') et les données sont définies au format ASCII sous la forme de couples ordonnés (attribut=valeur), séparés

par des caractères esperluette (‘&’). Un tel URL ressemblera alors à une chaîne de caractères comme celle présentée dans la *Figure 10*.

```
http://www.qqch.fr/forum/index.php3?cat=1&page=2
```

Figure 10. Un URL avec des paramètres

Cet information représente que au script appelé « index.php » seront envoyés les données « cat » avec une valeur « 1 », et « page » avec une valeur « 2 ».

II.3.1.2. Les hyperliens

Les hyperliens sont une partie essentielle de tout projet hypertexte. Un hyperlien est défini comme étant un fragment de texte, une image, ou une région dans une image, qui est sensible, et qui joue le rôle d’un pointeur vers une autre ressource, en général une page HTML. Les hyperliens sont définis en utilisant la balise `<A>` (Anchor). L’attribut le plus important de cette balise est `HREF` (Hyperlink Reference) qui sert à spécifier l’URL du document que le navigateur doit rapatrier et présenter lorsque l’utilisateur clique sur ce lien. À titre d’exemple nous présentons la spécification présentée dans la *Figure 11*.

```
<A HREF="http://www.laas.fr">La page du LAAS-CNRS</A>
```

Figure 11. Définition d’un hyperlien avec un URL absolu

Dans ce cas, lorsque l’utilisateur clique sur le texte « La page du LAAS-CNRS », son navigateur chargera et présentera la ressource qui se trouve à l’URL « http://www.laas.fr ».

Quand on spécifie un URL en utilisant cet élément, l’adresse dans l’attribut HREF peut être absolue ou relative. Dans le cas de l’exemple précédent il s’agit une adresse absolue. Une adresse relative se caractérise par le fait qu’elle ne contient ni protocole, ni nom du serveur, ni numéro de port, mais uniquement un chemin d’accès à la ressource. Dans un tel cas au moment que l’utilisateur clique sur ce lien, le navigateur fait l’hypothèse que ces trois paramètres sont les mêmes que ceux de l’URL de la ressource actuellement chargée dans le navigateur, et au moment où l’utilisateur clique sur ce lien, le navigateur ajoute automatiquement cette information à l’URL relatif, devenant ainsi un URL absolu avant d’envoyer la requête au serveur. À titre d’exemple considérons le fragment de code de la *Figure 12*.

```
<A HREF="/laasfr/index.html">L’index en Français</A>
```

Figure 12. Définition d’un hyperlien avec un URL relatif

Dans ce cas, si l’URL de la ressource dans laquelle se trouve l’hyperlien est « http://www.laas.fr », et que l’utilisateur clique sur cet hyperlien, son navigateur présentera la ressource dont l’URL absolu est « http://www.laas.fr/laasfr/index.html ».

Les conventions utilisées pour la spécification des chemins d’accès des URLs suivent le même standard que la plupart des systèmes de exploitation :

- un chemin absolu commence para un caractère ‘/’ ;
- la chaîne de caractères ‘..’ représente le répertoire de niveau immédiat supérieur au répertoire actuel.

Une caractéristique de HTML est qu'il permet la définition des ancres à l'intérieur d'une page HTML. Ensuite, il est possible de définir des hyperliens vers de telles ancres pour faire un saut exactement à l'endroit du fichier que l'on veut que soit affiché (là où se trouve l'ancre). Le lien peut se trouver dans le même fichier HTML, ou bien dans un autre fichier HTML. Dans ce dernier cas, une fois que la ressource a été complètement chargée dans le navigateur, un saut s'effectue à l'intérieur de la page jusqu'à l'endroit où l'ancre a été définie. Les liens ce de type sont appelés IPL (de l'anglais Intra-Page Link – liens à l'intérieur d'une page).

Dans ce contexte on utilise l'élément `<A>` deux fois : la première pour définir le nom du point d'ancrage vers lequel on souhaite créer un hyperlien par utilisation d'un attribut `NAME` de cette balise, et la deuxième pour spécifier l'hyperlien lui-même. L'exemple de la *Figure 13* illustre cette approche.

```
<A NAME="exemples">
. . .
<A HREF="#exemples">Cliquez ici pour voir quelques exemples</A>
```

Figure 13. Définition d'un IPL

Dans ce fragment de code, la première ligne dénote la création d'un point d'ancrage appelé « exemples », alors que la seconde dénote l'hyperlien qui pointe vers le point d'ancrage précédemment spécifié. Notons que dans la spécification d'un hyperlien de ce type, le nom du point d'ancrage doit être précédé du caractère dièse ('#'). Le cas où l'IPL se trouve dans un autre fichier est illustré dans l'exemple de la *Figure 14*.

```
http://www.laas.fr/laasfr/index.html#intranet
```

Figure 14. Un URL avec une définition d'ancre

De la même façon qu'il est possible de définir des hyperliens sur des fragments de texte, il est également possible d'utiliser des images. Il suffit alors d'indiquer une référence à une image à l'intérieur d'une balise `<A>`, comme cela est illustré dans l'exemple de la *Figure 15*.

```
<A HREF="/laasfr/index.html"> <IMG SRC="photo_laas.gif"> </A>
```

Figure 15. Définition d'un hyperlien par l'utilisation d'une image en HTML

Une troisième manière de créer des hyperliens est par l'utilisation d'images traitées en tant que cartes sensibles, de telle façon que, selon la partie de l'image où l'utilisateur clique, sa navigation pourra être envoyée vers un URL différent. Ceci est réalisable par l'utilisation des éléments HTML `<IMG>` et `<MAP>`, comme nous pouvons le voir dans l'exemple de la *Figure 16*.

```
<IMG SRC="map.gif" USEMAP="#navigation_map">
. . .
<MAP NAME="navigation_map">
   <AREA SHAPE="RECT" COORDS="23, 47, 58, 68" HREF="accueil.html">
   <AREA SHAPE="CIRCLE" COORDS="120, 246, 150, 246" HREF="infos.html">
. . .
</MAP>
```

Figure 16. Définition d'hyperliens par l'utilisation de cartes sensibles

II.3.1.3. Les formulaires

HTML donne la possibilité d'établir des formulaires grâce à l'élément `<FORM>`. Dans des formulaires l'utilisateur peut compléter des champs de saisie, dans des champs texte entrer plusieurs ligne de texte, faire des choix dans des listes et cliquer sur des boutons. Quand le formulaire est rempli complètement, l'utilisateur peut cliquer sur un bouton pour envoyer le formulaire. Habituellement les données saisies sont traitées par un programme CGI sur le serveur. Une fois les données traitées, le CGI engendre une réponse, qui est envoyée au navigateur de l'utilisateur qui a envoyé la requête. Tout formulaire HTML comprend au moins trois types de composants : un en-tête du formulaire, des champs de saisie ou de sélection et finalement des boutons de commande.

Des formulaires peuvent avoir de multiples fonctions. Ainsi on les emploie par exemple :

- pour récolter de l'utilisateur des renseignements déterminés ayant la même structure ;
- pour permettre aux utilisateurs la recherche dans des bases de données ;
- pour donner aux utilisateurs la possibilité de donner leur quote-part de données à une base de données ;
- pour offrir à l'utilisateur la possibilité d'une interaction individuelle par exemple en commandant un produit déterminé dans un assortiment de produits.

Dans la définition d'un formulaire il est possible de définir la méthode à utiliser pour envoyer la requête au serveur, l'URL qui indique l'adresse où se trouve le programme qui devra traiter les données envoyées, et un ensemble champs où l'utilisateur devra saisir l'information que lui est demandée. À titre d'exemple, considérons le formulaire HTML présenté dans la *Figure 17*.

Figure 17.Un formulaire HTML simple

Ce formulaire, qui contient trois champs de saisie, deux champs de sélection et deux boutons de commande, est spécifié par le code HTML présenté dans la *Figure 18*.

```
<FORM ACTION="http://www.monsite.fr/prog/adduser.cgi" METHOD="post">
  <P>
    Prénom: <INPUT TYPE="text" NAME="prenom"><BR>
    Nom: <INPUT TYPE="text" NAME="nom"><BR>
    Courriel: <INPUT TYPE="text" NAME="courriel"><BR>
    <INPUT TYPE="radio" NAME="sex" VALUE="Homme"> Homme<BR>
    <INPUT TYPE="radio" NAME="sex" VALUE="Femme"> Femme<BR>
    <INPUT TYPE="submit" VALUE="Envoyer">
    <INPUT TYPE="reset" VALUE="Réinitialiser">
  </P>
</FORM>
```

Figure 18.Définition d'un formulaire simple en HTML

La première ligne de ce fragment de code HTML correspond à l'en-tête du formulaire, qui spécifie où les données saisies au moyen de ce formulaire devront être traitées ; dans ce cas, il s'agit d'un CGI dont l'URL est http://www.monsite.fr/prog/adduser, et la méthode à utiliser pour envoyer les données au serveur est « POST ».

À l'intérieur de l'élément `<FORM>` apparaissent une série d'éléments `<INPUT>`. Les cinq premières correspondent à des champs de saisie ou de sélection. Les trois premières correspondent à des champs de type « TEXT », et les deux dernières à des boutons. Ces balises peuvent être associées à plusieurs attributs dont les plus utilisés sont « TYPE », pour indiquer le type de l'entrée (TEXT, PASSWORD, CHECKBOX, RADIO, etc), et « NAME » pour indiquer le nom d'un champ. Il est également possible d'indiquer une valeur par défaut en utilisant l'attribut « VALUE ».

Finalement les deux dernières balises `<INPUT>` correspondent à des boutons de commande. Le bouton de type « submit » permet d'envoyer les données au serveur, et le bouton de type « reset » permet de réinitialiser le formulaire.

II.3.1.4. Les cadres

A l'aide de cadres, qui sont disponibles à partir de la version 4 de HTML, il est possible de diviser l'affichage d'un navigateur en différentes parties définissables librement. Chaque partie peut avoir son propre contenu. Les différentes parties de l'affichage (appelées cadres) peuvent avoir un comportement statique (i.e. non scrolling regions – régions sans possibilité de défilement) ou un comportement dynamique. Il est aussi possible de définir si les cadres peuvent être redimensionnables par l'utilisateur. Les liens dans un cadre peuvent appeler des fichiers qui sont ensuite affichés dans un autre cadre. Une page Web contenant des cadres est présentée dans la *Figure 19*.

Ici nous pouvons apprecier l'existence de trois cadres, chacun étant identifié par un nom. Le code HTML permettant de générer cette page est présenté, de façon condensée, dans la *Figure 20*.

Cette page a été rapatriée de « http://java.sun.com/j2se/1.5.0/docs/api/ ». Le fichier dans lequel se trouve la définition d'une page contenant des cadres est une page Web, qui ne contient pas d'élément `<BODY>`, mais l'ensemble des définitions de cadres introduit par l'élément `<FRAMESET>`.

Chaque définition de cadre doit inclure l'attribut « SRC » par lequel on indique où se trouve la ressource qui devra être affichée dans ce cadre. Il est en général conseillé de donner un nom à chaque cadre par l'utilisation de l'attribut « NAME », mais ce n'est pas indispensable. Ce nom sera utilisé pour adresser, dans le contexte de la définition des hyperliens, l'affichage des ressources dans des cadres spécifiques parmi ceux disponibles dans l'ensemble des cadres.

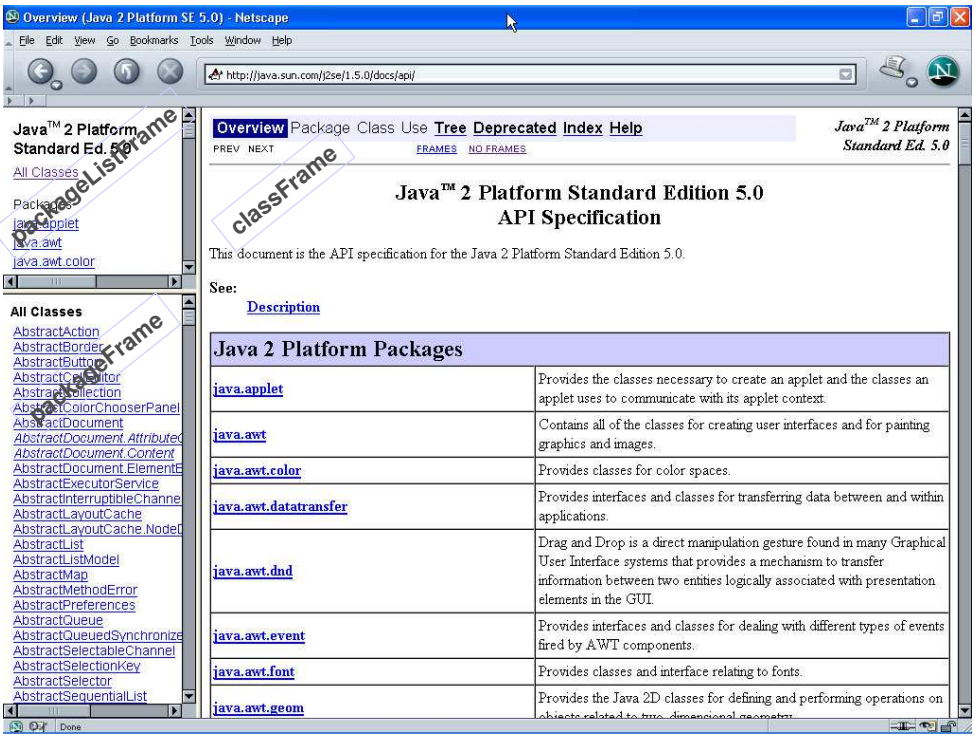


Figure 19.Une fenêtre de navigation avec des cadres

```
<FRAMESET COLS="20%,80%">
  <FRAMESET ROWS="30%,70%">
    <FRAME SCR="overview-frame.html" NAME="packageListFrame">
    <FRAME SCR="allclasses-frame.html" NAME="packageFrame">
  </FRAMESET>
  <FRAME SCR="overview-summary.html" NAME="classFrame">
</FRAMESET>
```

Figure 20. Définition d'un ensemble de cadres

II.3.1.5. Les paramètres de destination des fenêtres de navigation

Habituellement lorsqu'un utilisateur clique sur un hyperlien, la ressource rapatriée est présentée dans la même fenêtre du navigateur ou, le cas échéant, dans le même cadre. Cependant il existe la possibilité de spécifier un comportement différent au moyen d'un attribut « TARGET » présent dans les éléments `<A>` et `<FORM>`. Les valeurs possibles acceptées par cet attribut sont les suivantes :

- **_blank** : pour ouvrir la ressource indiquée dans une nouvelle fenêtre du navigateur.
- **_parent** : pour ouvrir la ressource indiquée dans le cadre de niveau immédiatement supérieur de celui où l'hyperlien est spécifié; cette valeur est normalement utilisée dans le cas de cadres imbriqués à plusieurs niveaux.
- **_self** : pour ouvrir la ressource indiquée exactement dans la même fenêtre ou cadre où se trouve l'hyperlien, ce qui équivalent à ne pas utiliser l'attribut « TARGET ».
- **_top** : pour ouvrir la ressource indiquée dans le cadre de plus haut niveau, c'est-à-dire au niveau de la fenêtre principale de navigation, éliminant ainsi toute structure de cadre existant.
- **nom-de-cadre** : pour ouvrir la ressource dans le cadre spécifié par « nom-de-cadre ».

II.3.2. Le protocole HTTP

Le protocole de communication HTTP (Hyper Text Transfer Protocol) [HTTP] est le protocole utilisé par définition pour le rapatriement de ressources dans le Web. Un client, tel qu'un navigateur Web, envoie une requête à un serveur qui retourne une réponse terminant ainsi la transaction. La version actuelle du protocole, HTTP 1.1, est décrite dans le RFC 2616 [RFC2616], mais la version 1.0, décrite dans la RFC 1945 [RFC1945] est encore très utilisée notamment pour des serveurs Proxy.

Ce protocole s'appuie sur des connexions TCP et le serveur est associé par défaut au port 80 (d'autres ports libres peuvent cependant être utilisés). Une caractéristique importante de HTTP est qu'il est un protocole en mode texte, et donc lisible par un humain, et facile à déboguer.

II.3.2.1. Le mode de fonctionnement général du protocole HTTP

Le protocole HTTP est un protocole sans état, ce qui signifie que lorsqu'un utilisateur envoie une requête à un serveur, et que ce serveur a satisfait cette requête, ou a le cas échéant retourné un message d'erreur, la connexion est terminée et il ne reste plus aucune trace de cette opération lors de l'envoi d'une prochaine requête.

Dans la définition de la version 1.0, il est établi que, par défaut, pour chaque ressource principale (par exemple, chaque page Web), ainsi que pour chaque composant inclus dans une ressource principale (par exemple une image), il faut ouvrir une connexion TCP spécifique. Ainsi, dans le cas d'une page Web contenant quatre cadres et cinq images par cadre, le client devrait ouvrir 21 connexions TCP, ce qui crée une surcharge évidente liée à la création et la fermeture de ces connexions TCP et peut ainsi provoquer un ralentissement de la transmission des ressources associées à cette page Web. Nous présentons dans la *Figure 21* une illustration de ce mode de fonctionnement.

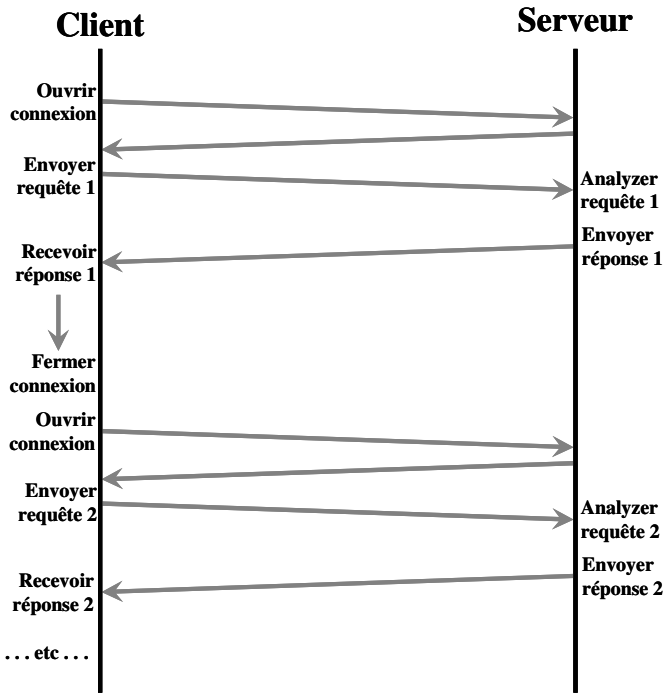


Figure 21.Rapatriement de ressources sur des connexions TCP non persistantes

Il est cependant possible de maintenir la connexion principale ouverte jusqu'à ce que toutes les ressources liées à une page Web soient transmises, c'est à dire la ressource

principale ainsi que toutes les ressources associées. C'est ce que l'on appelle des connexions persistantes. Dans HTTP 1.1, ce mode de fonctionnement est devenu le mode standard. Ainsi, si nous souhaitons que chaque ressource soit transmise par une connexion TCP distincte, nous devons l'indiquer explicitement au niveau du serveur. Cette notion est illustrée dans la *Figure 22*.

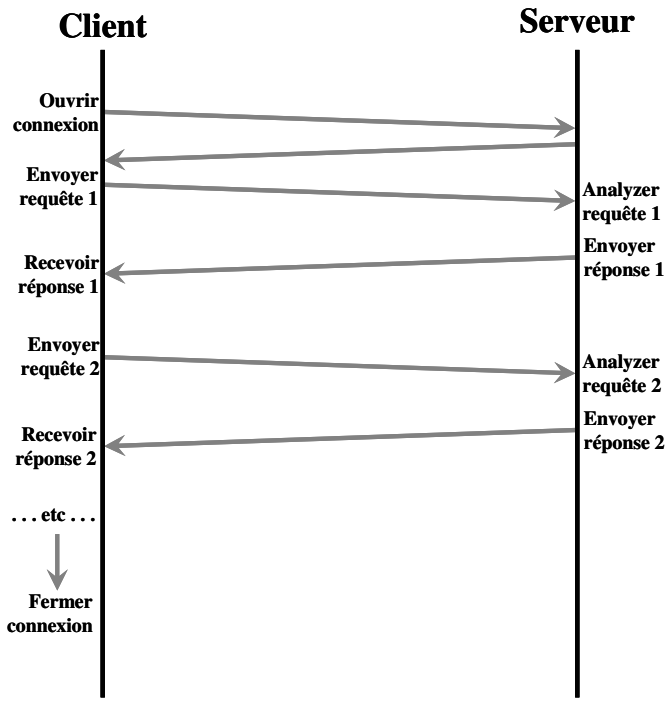


Figure 22.Rapatriement de ressources sur une connexion persistante

Lors de l'utilisation de serveurs proxy pour accéder à des ressources, il est fortement conseillé de ne pas utiliser de connexions persistantes, mais de traiter le rapatriement de chaque ressource associée à une page Web par l'utilisation d'une connexion TCP distincte.

II.3.2.2. Les en-têtes HTTP

Tous les échanges d'information entre un client et un serveur, par le biais de requêtes ou de réponses, contiennent des en-têtes. Ces en-têtes sont utilisés pour véhiculer des informations au serveur auquel la requête a été envoyée, ou au client qui a fait la requête. De manière générale, nous avons quatre types d'en-têtes :

- Des en-têtes généraux : ils peuvent apparaître aussi bien dans des requêtes que dans des réponses, et servent à spécifier des caractéristiques générales liées à une opération de rapatriement en particulier ;
- Des en-têtes de requête : ce sont des en-têtes spécifiques aux requêtes, permettant au serveur d'adapter sa réponse par rapport à des caractéristiques de l'utilisateur ou du navigateur qu'il utilise pour accéder aux ressources ;
- Des en-têtes de réponse : ce sont des en-têtes spécifiques aux réponses, permettant au serveur de donner au client des informations concernant les caractéristiques de la réponse elle-même ;
- Des en-têtes d'entité : ce sont des en-têtes HTTP qui décrivent le contenu du corps d'une requête ou d'une réponse.

  Les requêtes HTTP ont la structure générale présentée dans la *Figure 23*.

```
METHODE URL HTTP/version
... En-têtes généraux ...
... En-têtes de requête ...
... En-têtes d'entité (optionnels) ...
                                                ⇦ Ligne vide
```

Figure 23. Forme générale des requêtes HTTP

Les réponses HTTP ont la structure générale présentée dans la *Figure 24*.

```
HTTP/version code-d'état message
... En-têtes généraux ...
... En-têtes de réponse ...
... En-têtes d'entité (optionnels) ...
                                                ⇦ Ligne vide
... Ressources (éventuellement) ...
```

Figure 24. Forme générale des réponses HTTP

Au delà des en-têtes standards définis dans la spécification de HTTP, des applications spécifiques peuvent ajouter des en-têtes ad-hoc, ce qui permet d'étendre les capacités d'opération de ce protocole.

Nous présentons maintenant une description sommaire de quelques uns des en-têtes les plus représentatifs, leur description détaillée étant au delà de la portée de ce document.

- ***En-têtes généraux :***
  - **`Cache-Control`** : cet en-tête est utilisé pour contrôler différents aspects de la mise en cache des ressources rapatriées ;
  - **`Connection`** : cet en-tête sert à spécifier des options de communication pour la connexion, comme par exemple, le choix d'utiliser ou non des connexions TCP persistantes (par exemple keep-alive ou close) ;
  - **`Date`** : cet en-tête indique la date et l'heure à laquelle le message qui contient cet en-tête a été engendré.
- ***En-têtes de requête :***
  - **`Accept`** : cet en-tête indique au serveur quels types de média sont acceptables par le client. Les types acceptables sont définis en tant que types MIME (ce qui sera abordé ultérieurement dans cette section) ;
  - **`Accept-Charset`** : cet en-tête indique au serveur les jeux de caractères (charsets) acceptables par le client (par exemple ISO-8859-5) ;
  - **`Accept-Encoding`** : cet en-tête indique au serveur les types de codage acceptés par le client (par exemple ressources compressées) ;
  - **`Accept-Language`** : cet en-tête indique au serveur la langue préférée de l'utilisateur, telle qu'elle a été définie dans le navigateur ;
  - **`If-Modified-Since`**, **`If-Unmodified-Since`**, **`If-Match`**, **`If-None-Match`** : ces en-têtes sont utilisés pour faire des requêtes conditionnelles, de manière à ne rapatrier une ressource du serveur d'origine que si elle est plus récente que celle stockée dans le système de cache du navigateur ;
  - **`Referer`** : cet en-tête indique au serveur l'URL du document à partir duquel la requête a été envoyée ;
  - **`User-Agent`** : cet en-tête indique au serveur le type du navigateur (nom et version) à partir duquel la requête a été engendrée.

- ***En-têtes de réponse :***
  - **`Location`** : cet en-tête indique au client l'URL d'une ressource qu'il a demandé, et qui ne se trouve plus à l'endroit où le client l'a spécifié dans sa requête (voir les codes de réponse ultérieurement dans ce chapitre) ;
  - **`Server`** : cet en-tête indique au client le type de serveur qui a engendré une réponse donnée.
- ***En-têtes d'entité :***
  - **`Content-Base`** : cet en-tête indique au client l'URL qui servira de base à la définition relative des hyperliens.
  - **`Content-Encoding`** : cet en-tête indique au client le type de codage utilisé lors de l'envoi d'une ressource (par exemple, gzip, compress).
  - **`Content-Language`** : cet en-tête indique au client le langage (anglais, français, espagnol, etc.) dans lequel la ressource rapatriée est rédigée.
  - **`Content-Length`** : cet en-tête indique au client la longueur en octets de la ressource qu'il est en train de lui envoyer.
  - **`Content-Type`** : cet en-tête indique au client le type MIME de la ressource qu'il est en train d'envoyer (par exemple, texte, image, etc.).
  - **`Expires`** : cet en-tête indique au client la date et l'heure d'expiration de la ressource, dans le but que la copie locale gardée dans le cache du navigateur ne soit plus utilisée au-delà de cette date d'expiration.
  - **`Last-Modified`** : cet en-tête indique au client la date et l'heure de création ou de dernière modification de la ressource, dans le but que la copie locale gardée dans le cache du navigateur ne soit plus utilisé après cette date de dernière mise à jour.

II.3.2.3. Les types MIME

Le standard MIME (Multipurpose Internet Mail Extensions – extension de courrier Internet à usages multiples), défini dans les RFC 2045 à 2049 [RFC2045]-[RFC2049], constitue un moyen pour spécifier différents types de medias qui peuvent être échangés par courrier électronique. Ce standard correspond à une amélioration du standard RFC 822 [RFC0822] qui a été défini à l'origine pour envoyer des messages en format texte de base. MIME a également été adopté par le protocole HTTP et est utilisé notamment dans l'en-tête de requête « Accept » et dans l'en-tête d'entité « Content-Type ». Le premier cas permet au client d'indiquer au serveur les types de données qu'il peut traiter, et dans le deuxième cas d'indiquer au client le type d'une ressource donnée.

La spécification d'un type de média comprend deux parties: un type principal, qui spécifie la catégorie générale d'une ressource, et un sous-type, qui décrit le type spécifique d'une ressource. Nous présentons dans la *Figure 25* les principales catégories de types de medias, avec quelques exemples représentatifs.

La définition des types MIME est en constante évolution, et quiconque peut définir un nouveau type selon ses besoins. Une requête de création d'un nouveau type de média MIME doit être enregistrée auprès de l'IANA (Internet Assigned Numbers Authority – autorité des numéros assignés d'Internet) [IANA].

Type général	Type spécifique
application/*	application/pdf application/postscript application/rtf
audio/*	audio/basic audio/mpeg
image/*	image/gif image/jpeg image/png image/tiff
message/*	message/http message/news message/rfc822
model/*	model/vrml
multipart/*	multipart/encripted multipart/form-data multipart/mixed
text/*	text/enriched text/html text/plain text/richtext text/rtf text/xml
video/*	video/jpeg video/mpeg video/quicktime

Figure 25. Quelques types MIME parmi les plus utilisés

II.3.2.4. Les requêtes HTTP

Dans le contexte de la navigation sur le Web, toute interaction est déclenchée par un client, par le biais de son navigateur. De son côté, le serveur est en attente de l'arrivée des requêtes, pour les traiter et envoyer ensuite en retour au client les réponses pertinentes.

II.3.2.4.1. Les méthodes de requête

Du côté client, plusieurs méthodes existent pour lui permettre d'envoyer une requête à un serveur. Ces méthodes sont les suivantes :

- **`GET`** : cette méthode est utilisée pour demander au serveur d'envoyer en réponse la ressource spécifiée par l'URL indiquée. Si jamais des données doivent être envoyées au serveur dans la requête, ces données sont codées en tant que paramètres et envoyées dans l'URL même ;
- **`POST`** : cette méthode sert à demander à un serveur de satisfaire une requête, qui sera nécessairement paramétrée par certaines données définies par l'utilisateur au moyen d'un formulaire dans une page Web. Le contenu de la réponse sera en fonction de ces données ;
- **`HEAD`** : cette méthode réalise la même action que la méthode GET, mais dans la réponse obtenue, seuls les en-têtes de la réponse sont rapatriées, et non pas la réponse complète ;
- **`PUT`** : cette méthode est utilisée pour envoyer au serveur et lui demander de l'archiver, la ressource indiquée par l'URL. Elle reste très peu utilisée, et pas tous les serveurs ne l'implémentent ;
- **`DELETE`** : cette méthode est utilisée pour supprimer dans le serveur la ressource indiquée par l'URL. De la même manière que la méthode PUT, elle reste très peu utilisée.

Parmi toutes ces méthodes, les plus utilisées sont donc « GET », « POST » et « HEAD ». La méthode « GET » est la plus utilisée dans le monde du Web pour rapatrier

des ressources. Dans sa forme la plus simple elle se présente de la manière présentée dans la *Figure 26*.

```
GET /index.html HTTP/1.1
Host: www.laas.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804
Netscape/7.2 (ax)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Figure 26. Requête HTTP (« GET ») sans envoi de données

Habituellement les ressources demandées au moyen de cette méthode sont des pages Web classiques, qui n'envoient en général pas de données dans la requête. Dans ce cas, les données capturées au moyen du formulaire sont envoyées au serveur comme faisant partie intégrale de la ressource demandée. Prenons par exemple le cas du formulaire présenté dans la *Figure 17*. Si la méthode de traitement est définie comme étant « GET », dès que l'utilisateur clique sur le bouton « Envoyer », la requête présentée dans la *Figure 27* est transmise au serveur.

```
GET /prog/adduser.cgi?prenom=Guillermo+de+Jesus&nom=HOYOS-RIVERA&
courriel=ghoyos%40laas.fr&sex=homme HTTP/1.1
Host: www.laas.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804
Netscape/7.2 (ax)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Figure 27. Requête HTTP (GET) avec envoi de données

Nous pouvons ici noter clairement comment les données saisies dans un formulaire apparaissent dans la requête, selon un standard prédéfini. En premier lieu, nous distinguons la chaîne de caractères que représente le chemin d'accès au programme traitant les données, et la chaîne de caractères contenant les données elles-mêmes, ce qui est réalisé au moyen du caractère '?'. Des contraintes de type sont de plus imposées aux données. Ainsi les caractères autorisés sont les caractères alphanumériques [0-9a-zA-Z] et quelques caractères spéciaux tel que '$', '-', '_', '.', '+', '!', '*', ''', '(' et ')'.

Dans le cas de requêtes envoyées par la méthode « POST » le traitement est différent, cette méthode ayant été conçue pour « poster » des données sur un serveur. Même si la méthode « GET » permet l'envoi de données, elle souffre de deux inconvénients majeurs :

- Les données sont envoyées en étant codées directement dans la requête, et donc visibles au niveau d'affichage de l'URL dans le navigateur de l'utilisateur, ce qui dans certains cas peut être indésirable.
- La longueur de la chaîne de données est limitée, en général à 1024 octets, ce qui poser un problème lorsque la taille des données à envoyer dépasse cette limite.

Lorsqu'on utilise la méthode « POST », les données saisies dans le formulaire sont envoyées dans un paquet TCP distinct sur la même connexion. Ainsi, le même formulaire que dans l'exemple précédent, engendre la requête HTTP présentée dans la *Figure 28*.

Paquet n

```
POST /prog/adduser.cgi HTTP/1.1
Host: www.laas.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804
Netscape/7.2 (ax)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 78
```

Paquet n+1

```
prenom=Guillermo+de+Jesus&nom=HOYOS-RIVERA&courriel=ghoyos%40laas.fr& sex=homme
```

Figure 28. Requête HTTP (POST) avec envoi de données

La méthode « HEAD » a quasiment le même comportement que la méthode « GET », sauf que dans ce cas le serveur n'enverra au client que les en-têtes de la réponse, et non pas son corps. Cette méthode est particulièrement utile lorsque l'on veut uniquement tester l'accessibilité d'une ressource sans nécessairement devoir la rapatrier.

Finalement les méthodes « PUT » et « DELETE » ne sont pas supportées, par défaut, par les serveurs Web couramment accessibles. Ces méthodes ne seront pas détaillées ici, car elles ne présentent pas d'intérêt dans le contexte de navigation coopérative que nous proposons.

II.3.2.5. Les réponses HTTP

En présence d'une requête HTTP, le serveur réagit toujours, soit en envoyant la ressource demandée au client, soit en envoyant un message d'information ou d'erreur, permettant au client de diagnostiquer pourquoi la requête n'a pu être satisfaite. Les réponses HTTP ont un format similaire à celui des requêtes et sont constituées d'une première ligne contenant le code d'état de la réponse, suivi d'un ensemble d'en-têtes et, le cas échéant, du corps de la réponse.

Cette première ligne de la réponse comprend trois parties. La première indique au client la version du protocole HTTP utilisée pour transmettre la réponse, la seconde informe au client le code d'état de la réponse, et enfin la troisième donne un descriptif, à titre de diagnostic, qui n'est pas interprété par le navigateur, mais qui sert à l'utilisateur, dans les cas d'erreurs, pour comprendre la raison pour laquelle la requête a échoué.

Les codes d'état HTTP sont compris entre 100 et 599, et sont répartis en cinq catégories : 1xx – Informationnel, 2xx – Succès, 3xx – Redirection, 4xx – Erreur du client et 5xx – Erreur du serveur. Le premier chiffre détermine la signification générale du code d'état, et les chiffres suivants spécifient cette condition en détail. Nous présentons par la suite, dans les figures suivantes, une brève description des codes d'état les plus souvent trouvés, en indiquant si ces codes sont déjà définis pour la version 1.0 de HTTP.

Code	1.0	Message	Description
200	●	OK	La requête a été satisfaite avec succès, et l'information est retournée en réponse.
204	●	No Content	La réponse est intentionnellement vide, donc le navigateur ne doit pas changer le document actuellement affiché.
205		Reset Content	Indique au navigateur de réinitialiser le document actuel, ce qui est particulièrement utilisé pour remettre à blanc les champs de saisie d'un formulaire.

Figure 29. Codes d'état 2xx

Code	1.0	Message	Description
301	●	Moved Permanently	La ressource demandée a été re-localisée. L'endroit où on peut la trouver est indiqué dans l'en-tête de réponse « Location: ». Le navigateur devra alors rapatrier la ressource de ce nouvel URL.
302	●	Moved Temporarily	Ce code est similaire au 301, mais la re-localisation est considérée comme temporelle.
304	●	Not Modified	La version stockée dans le cache du navigateur est toujours à jour, donc la ressource n'est pas rapatriée.

Figure 30. Codes d'état 3xx

Code	1.0	Message	Description
400	●	Bad Request	La requête n'a pu être comprise par le serveur. Ce code est utilisé quand aucun des autres codes d'erreur ne peut s'appliquer.
401	●	Unauthorized	Ce code est envoyé au client tant que le serveur ne reçoit pas dans la requête des qualifications valables dans l'en-tête « Authorization: ». Dans la réponse un en-tête « WWW-Authenticate: » est inclus.
404	●	Not Found	La ressource demandée n'existe pas (plus).
407		Proxy Authentication Required	Similaire au code 401, mais celui-ci s'applique dans le cas de l'accès à un serveur Proxy.

Figure 31. Codes d'état 4xx

Code	1.0	Message	Description
500	●	Internal Server Error	C'est un code d'erreur générique indiquant qu'il est arrivé une erreur inattendue dans le serveur.
503	●	Service Unavailable	Le service est temporairement hors service.
505		HTTP Version Not Supported	Le protocole spécifié dans la requête ne peut pas être traité par le serveur.

Figure 32. Codes d'état 5xx

À partir de l'information obtenue du code d'état reçu, et éventuellement des en-têtes de la réponse, le client exécutera une action spécifique. Une réponse HTTP typique est présentée dans la *Figure 33*.

```
HTTP/1.1 200 OK
Date: Wed, 08 Dec 2004 13:11:02 GMT
Server: Apache/1.3.31 (Unix) DAV/1.0.3 PHP/4.3.8 mod_ssl/2.8.19 OpenSSL/0.9.7d
Content-Location: index.html.en
Vary: negotiate,accept-language
TCN: choice
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
Content-Language: en

. . . Corps de la réponse . . .
```

Figure 33. Une réponse HTTP avec succès

Ici nous pouvons voir le cas d'une réponse réussie, puisque le code d'état est 200. Par contre il peut également arriver qu'une ressource ne soit pas accessible, ce qui conduit à une réponse du type présenté dans la *Figure 34*.

```
HTTP/1.1 404 Not Found
Date: Wed, 08 Dec 2004 14:04:07 GMT
Server: Apache/1.3.31 (Unix) DAV/1.0.3 PHP/4.3.8 mod_ssl/2.8.19 OpenSSL/0.9.7d
Content-Location: 404.shtml.en
Vary: negotiate,accept-language
TCN: choice
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
Content-Language: en

. . . Message d'erreur optionnel . . .
```

Figure 34. Une réponse HTTP avec notification d'erreur

Un dernier exemple est celui d'une requête dont la réponse indique au client que la ressource n'est plus à l'endroit où elle a été cherchée, mais dans un URL différent. Ce cas est présenté dans la *Figure 35*.

```
HTTP/1.1 302 Found
Location: http://www.google.fr/
Content-Type: text/html
Server: GWS/2.1
Transfer-Encoding: chunked
Content-Encoding: gzip
Date: Wed, 08 Dec 2004 14:07:56 GMT
Cache-Control: private, x-gzip-ok=""
```

Figure 35. Une réponse HTTP de réexpédition

Nous pouvons voir clairement ici que, en plus du code d'état de la réponse, nous avons un en-tête « Location: » ; la réaction immédiate du navigateur devra donc être de rapatrier la ressource qui se trouve à l'URL indiqué dans cet en-tête.

Chapitre III

Spécification et vérification formelle du modèle de synchronisation

III.1. Introduction

Le but de notre proposition est d'offrir à des utilisateurs connectés à une même session de navigation coopérative toute une gamme de services leur permettant de synchroniser leurs activités de navigation.

Imaginons, par exemple, le cas de deux utilisateurs, *i* et *j*, connectés à la même session de navigation coopérative. Dans un premier temps, ces deux utilisateurs travaillent de manière asynchrone : chacun navigue librement sur les pages Web de son choix. Considérons maintenant le cas où l'utilisateur *j* décide de synchroniser sa navigation à celle de l'utilisateur *i*. À partir de ce moment, chaque fois que l'utilisateur *i* exécute une action de navigation, le même document apparaît dans le navigateur des utilisateurs *i* et *j*. L'utilisateur *j* suit ainsi la navigation de *i* et ne peut plus exécuter de sa propre initiative des requêtes de navigation.

Nous présentons dans ce chapitre le modèle de synchronisation associé à notre proposition de navigation coopérative. Nous définissons dans un premier temps le modèle de synchronisation de base, introduisant les principales actions permettant d'établir et de supprimer des relations de synchronisation entre utilisateurs d'une même session de navigation coopérative. Nous proposons ensuite différentes formalisations de ce modèle de synchronisation de base, en développant des modèles s'appuyant sur des automates étendus (vue locale des actions de synchronisation) et des réseaux de Petri (vue globale des actions de synchronisation). Le modèle global, à base de réseaux de Petri, nous sert ensuite de base pour une vérification formelle de notre modèle de synchronisation. Finalement, nous montrons, dans un dernier paragraphe, comment étendre notre modèle de synchronisation, pour rendre notre proposition de navigation coopérative la plus flexible possible.

III.2. Le modèle de synchronisation de base

Dans notre proposition de navigation coopérative, nous avons défini plusieurs actions de synchronisation permettant de manière dynamique d'établir et de détruire des relations de synchronisation entre utilisateurs dans le but de contrôler leur navigation sur le Web. Nous allons dans un premier temps montrer comment représenter des relations de synchronisation entre utilisateurs au sein d'une même session de navigation coopérative, puis détailler les différents actions de synchronisation définies dans notre modèle de synchronisation de base.

III.2.1. Le SDT

La structure de base permettant de caractériser l'état des relations de synchronisation d'un ensemble d'utilisateurs appartenant à une même session de navigation coopérative est appelée un SDT (en anglais, « Synchronization Dependency Tree » - arbre de dépendances de synchronisation).

Définition 1. Un SDT est un arbre dans lequel les nœuds représentent les utilisateurs connectés à une même session de navigation coopérative, et les arcs représentent les relations de synchronisation entre ces utilisateurs. L'existence d'un arc orienté entre un nœud 2 (nœud associé à l'utilisateur 2) et un nœud 1, où 2 est le fils de 1, caractérise que la navigation de l'utilisateur 2 est synchronisée à celle de l'utilisateur 1.

Définition 2. Un utilisateur est dit « asynchrone » si son nœud est racine d'un SDT, ce qui signifie que sa navigation n'est pas synchronisée à celle d'un autre utilisateur. Un utilisateur est dit « synchrone » si le nœud auquel il est associé appartient à un SDT sans en être la racine ; dans ce cas, la navigation de cet utilisateur est synchronisée à celle du nœud racine du SDT.

A partir de ces définitions et de la structure arborescente du SDT, et considérant un ensemble d'utilisateurs appartenant à une même session de navigation coopérative, nous pouvons déduire que plusieurs utilisateurs peuvent être synchronisés à un même utilisateur, mais qu'un utilisateur ne peut être synchronisé qu'à un seul autre utilisateur.

Un utilisateur peut être soit asynchrone soit synchrone. S'il est asynchrone, il n'est pour sa navigation contraint par aucune relation de synchronisation et peut donc naviguer de sa propre initiative (sélection d'hyperliens, entrée d'URLs, et le conséquent chargement de documents associés). S'il est synchrone, il est contraint pour sa navigation par la relation de synchronisation établie entre lui et l'utilisateur associé au nœud racine du SDT auquel il appartient, ce qui est illustré dans la *Figure 36*.

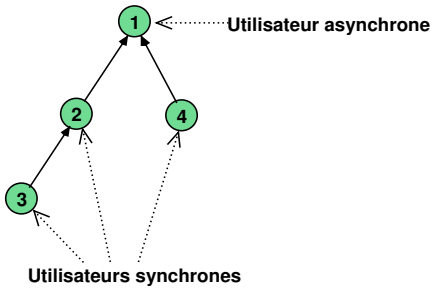


Figure 36. Utilisateurs synchrones et asynchrones

Dans cet exemple le seul utilisateur à pouvoir naviguer de sa propre initiative est l'utilisateur *1*, et à chaque fois que celui-ci exécute une action de navigation, les utilisateurs *2*, *3* et *4* doivent exécuter la même action de navigation. Nous pouvons voir qu'un SDT peut être constitué de plusieurs niveaux. Dans cet exemple, l'utilisateur *3* est synchronisé à l'utilisateur *2*, mais comme ce dernier est également synchronisé à l'utilisateur *1*, le résultat est que la navigation de l'utilisateur *3* est synchronisée à celle de l'utilisateur *1*.

Le nombre de SDTs qui peuvent exister à un instant donné dans une session de navigation coopérative est variable. Il dépend du nombre d'utilisateurs connectés à la session et de leur état de synchronisation, c'est à dire des relations de synchronisation établies entre eux. Si un utilisateur est asynchrone, et qu'aucun autre utilisateur n'est synchronisé avec lui, cela représente un SDT qui se réduit au nœud racine, dont le nœud de l'utilisateur est la racine. Ainsi le nombre maximum de SDTs correspond au nombre d'utilisateurs connectés à la session de navigation coopérative. Le nombre de SDTs varie

ensuite en fonction des actions de synchronisation effectuées par les utilisateurs et donc des relations de synchronisation établies entre les nœuds. Nous présentons dans la *Figure 37* quelques scénarios de synchronisation pouvant être associés à une même session de navigation coopérative.

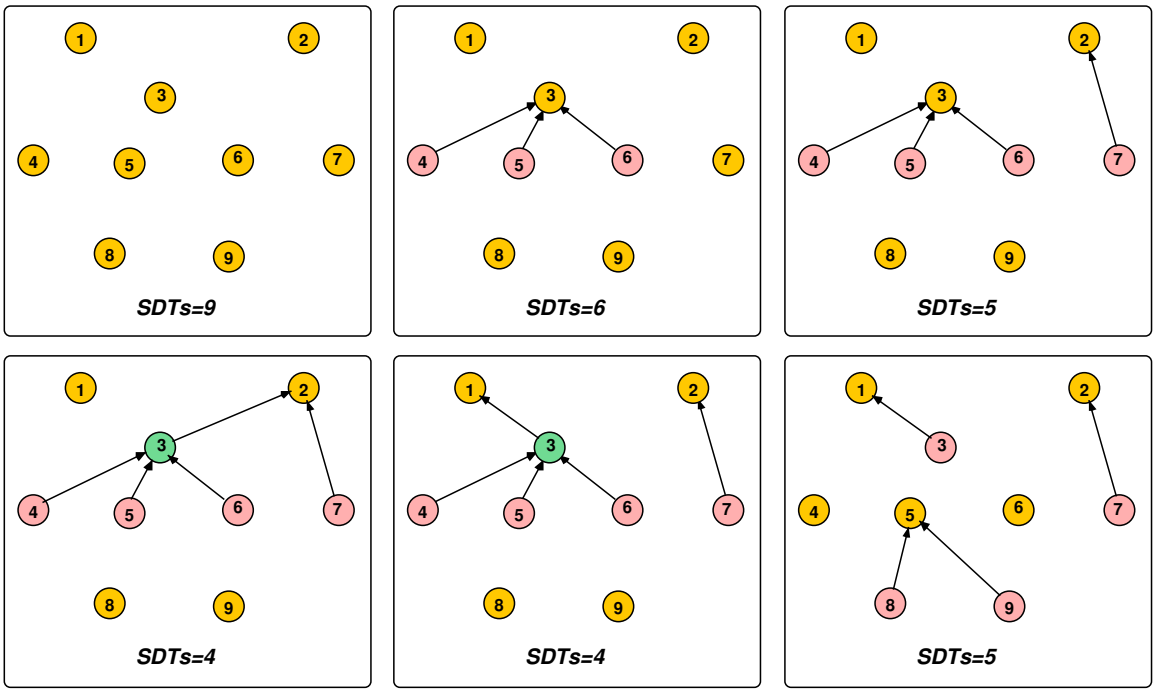


Figure 37. Evolution du nombre de SDTs dans une session de navigation coopérative

Les actions de synchronisation réalisées par des utilisateurs appartenant à une même session de navigation coopérative ont pour conséquence de modifier le nombre et la structure des SDTs associés à cette session. Ces actions de synchronisation se classent en deux catégories :

- des requêtes de synchronisation permettant aux utilisateurs de synchroniser leur navigation, créant ainsi des relations de synchronisation ;
- des requêtes de synchronisation permettant aux utilisateurs de se désynchroniser, supprimant ainsi des relations de synchronisation préalablement établies.

III.2.2. Création de relations de synchronisation

Les requêtes de synchronisation, qui aboutissent à la création de relations de synchronisation, sont du type « Follow ». Les deux implémentations de cette requête de synchronisation sont « I_Follow_You » et « You_Follow_Me ».

« I_Follow_You » exprime l'intention d'un utilisateur de se synchroniser à un autre utilisateur, et « You_Follow_Me » exprime l'invitation d'un utilisateur pour qu'un autre utilisateur se synchronise avec lui. La réalisation de ces requêtes de synchronisation est soumise à un mécanisme d'autorisation. Dans les deux cas, pour faire aboutir la requête et que la relation de synchronisation soit ainsi créée, il faut que l'utilisateur auquel la requête a été envoyée autorise de manière explicite la création de la relation de synchronisation.

En fonction des contraintes liées à la construction des SDTs, et étant donné qu'un utilisateur ne peut se synchroniser qu'à un seul autre utilisateur, la requête « I_Follow_You » ne peut être utilisée que pour créer des relations de synchronisation 1-1. Par contre, dans le cas de la requête « You_Follow_Me », il est possible d'envisager la

possibilité de lancer des invitations de synchronisation aussi bien à des utilisateurs individuels qu'à des groupes d'utilisateurs.

Chaque requête de synchronisation de type « I_Follow_You » conduit à composer deux SDTs (le nœud racine du SDT associé à l'utilisateur qui se synchronise devient le fils du nœud associé à l'utilisateur auquel il se synchronise). D'une manière similaire, chaque requête de synchronisation de type « You_Follow_Me » conduit à la composition de deux ou de plusieurs SDTs (nous nous restreindrons dans ce chapitre à la composition de deux SDTs uniquement pour toute requête de synchronisation « You_Follow_Me »).

A titre d'exemple, la *Figure 38* illustre la composition de deux SDTs suite à la création d'une relation de synchronisation (l'utilisateur *3* se synchronise à l'utilisateur *2*), ce qui peut être réalisé par un « I_Follow_You » à l'initiative de l'utilisateur *3*, ou d'un « You_Follow_Me » à l'initiative de l'utilisateur *2*.

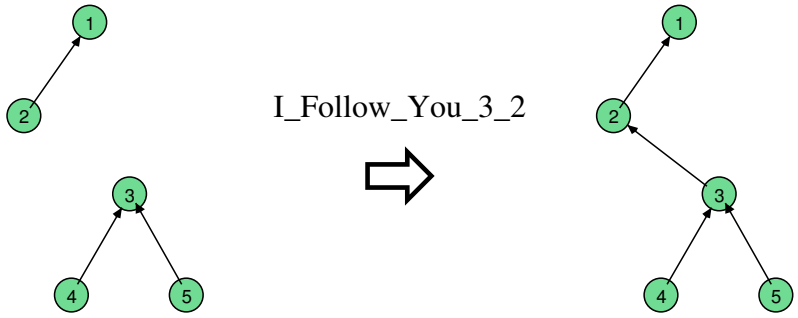


Figure 38. Illustration de la création d'une relation de synchronisation

III.2.3. Suppression de relations de synchronisation

La requête de synchronisation aboutissant à la suppression de relations de synchronisation préalablement établies est « I_Leave_You ». L'exécution de cette requête de synchronisation ne dépend d'aucun mécanisme d'autorisation préalable et est donc inconditionnelle. Par élimination de l'arc correspondant à la relation de synchronisation supprimée, cette requête a pour effet de décomposer un SDT en deux SDTs distincts.

A titre d'exemple, la *Figure 39* illustre la décomposition d'un SDT en deux SDTs suite à la suppression d'une relation de synchronisation établie préalablement entre l'utilisateur 2 et l'utilisateur 1, ce qui peut être réalisé par un « I_Leave_You » à l'initiative de l'utilisateur 2 ou de l'utilisateur 1.

Figure 39. Illustration de la suppression d'une relation de synchronisation

III.2.4. Synchronisation de la navigation

Lorsqu'un utilisateur exécute une action de navigation (appelée « Browse »), la ressource spécifiée est chargée dans le navigateur de l'utilisateur. Immédiatement après, l'URL de cette ressource est envoyée à tous les utilisateurs associés aux noeuds appartenant au même SDT de l'utilisateur qui vient d'exécuter l'action de navigation, et cette ressource est chargée automatiquement dans le navigateur de chacun de ces utilisateurs. Cette notion, à la base de notre proposition de navigation coopérative, est illustrée dans la *Figure 40*.

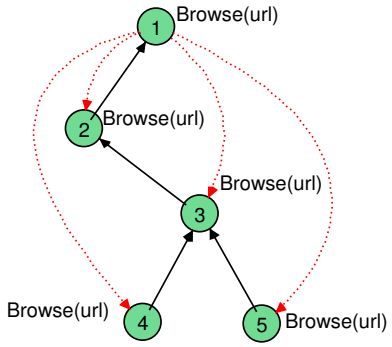


Figure 40. Illustration d'une navigation synchronisée

III.3. Formalisation du modèle de synchronisation de base

Nous proposons deux formalisations de notre modèle de synchronisation de base. La première s'appuie sur des automates étendus et son objectif est de décrire le comportement local d'un utilisateur pouvant recevoir et émettre des requêtes de synchronisation. La deuxième s'appuie sur des réseaux de Petri [Diaz-03] [Diaz-01] [Milner-84], et son objectif est de décrire le comportement global d'un ensemble d'utilisateurs créant et supprimant des relations de synchronisation. Cette deuxième formalisation servira de base à la vérification formelle de notre modèle de synchronisation de base qui sera présentée dans le paragraphe III.4.

Nous faisons l'hypothèse que le nombre d'utilisateurs au sein d'une session de navigation coopérative est connu et reste constant. Par souci de simplification, nous ne modélisons donc pas le fait que des utilisateurs puissent dynamiquement, alors que des relations de synchronisation sont déjà établies au niveau de la session, quitter ou joindre la session de navigation. Cette hypothèse sera levée au niveau de l'implémentation des mécanismes de synchronisation, comme nous le détaillerons dans le Chapitre V.

III.3.1. Formalisation par des automates étendus

Le but de cette modélisation est de décrire le comportement local d'un utilisateur qui peut recevoir et émettre les requêtes de synchronisation que nous avons décrites dans le paragraphe III.2. Nous choisissons comme formalisme un modèle d'automate, étant donné que nous désirons représenter le comportement séquentiel d'un utilisateur faisant partie d'une session de navigation coopérative. Ce modèle d'automate est étendu car nous représentons l'ensemble des SDTs associés à cette session sous la forme d'une structure de données à laquelle nous pouvons accéder par un ensemble de prédicats et actions. Nous représentons également la communication de l'utilisateur modélisé avec d'autres utilisateurs de la même session sous la forme de prédicats (réception d'une requête de

synchronisation ou réception d'une réponse à une requête précédente) ou d'actions (émission d'une requête de synchronisation ou émission d'une réponse à une requête précédente).

- *Requêtes de synchronisation :*
  - I_Follow_You : un utilisateur demande de se synchroniser à un autre utilisateur ;
  - You_Follow_Me : un utilisateur invite un (ou plusieurs) utilisateur(s) à se synchroniser avec lui ;
  - I_Leave_You : un utilisateur quitte la relation de synchronisation qu'il avait avec un autre utilisateur.
- *Réponses à des requêtes de synchronisation :*
  - I_Accept : Un utilisateur accepte une requête de synchronisation de type « Follow » ;
  - I_Abort : un utilisateur annule une requête de synchronisation de type « Follow », avant que celle-ci ne soit acceptée ou refusée ;
  - I_Refuse : Un utilisateur refuse une requête de synchronisation de type « Follow ».
- *Prédicats et actions de communication entre utilisateurs :*
  - I ? A : l'action de synchronisation A (requête ou réponse) est reçue de l'utilisateur i ;
  - I ! A : l'action de synchronisation A (requête ou réponse) est envoyée à l'utilisateur i.
- *Prédicats sur la structure de données caractérisant l'ensemble des SDTs de la session:*
  - async(i) : $U \rightarrow \{T, F\}$ est un prédicat qui donne vrai si l'utilisateur i est asynchrone, et que le nœud qui lui est associé est donc racine d'un SDT. Il a la possibilité de naviguer de sa propre initiative ;
  - sync(i, j) : $U \times U \rightarrow \{T, F\}$ est un prédicat qui donne vrai si l'utilisateur i est synchronisé à l'utilisateur j, la navigation de i suit donc celle de j ;
  - root(i, j) : $U \times U \rightarrow \{T, F\}$ est un prédicat qui donne vrai si le nœud associé à l'utilisateur i appartient au SDT dont la racine est le nœud associé à l'utilisateur j.
- *Fonctions sur la structure de données caractérisant l'ensemble des SDTs de la session:*
  - updateSDT-add(i, j) : cette fonction actualise l'ensemble des SDTs associés à la session de navigation coopérative, suite à l'ajout d'une relation de synchronisation entre les utilisateurs i et j (la navigation de i suit celle de j) ;
  - updateSDT-remove(i, j) : cette fonction actualise l'ensemble des SDTs associés à la session de navigation coopérative, suite à la suppression d'une relation de synchronisation entre les utilisateurs i et j (i devient ainsi asynchrone et peut alors naviguer de sa propre initiative).

Partant de ces notations, nous obtenons les deux automates étendus suivants : le premier automate étendu (*Figure 41*) présente le comportement local d'un utilisateur *i* qui se synchroniser à un autre utilisateur *j* (réception de la requête « You_Follow_Me » ou émission de la requête « I_Follow_You ») ainsi que le traitement des requêtes de désynchronisation associées « I_Leave_You ». Le deuxième automate étendu (*Figure 42*) présente le comportement local d'un utilisateur *i* avec qui un autre utilisateur *k* souhaite se synchronise ainsi que le traitement des requêtes de désynchronisation associées « I_Leave_You ».

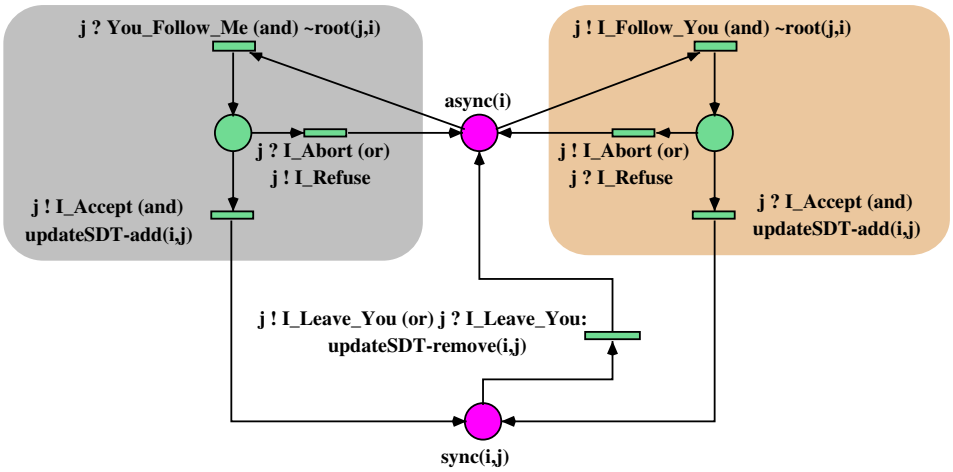


Figure 41. Synchronisation de l'utilisateur i avec un utilisateur j (modèle de synchronisation de base)

Nous pouvons voir clairement les deux états majeurs possibles dans lesquels l'utilisateur *i* peut se trouver par rapport à l'utilisateur *j* : *i* peut être asynchrone (état *async(i)*) ou synchronisé à *j* (état *sync(i, j)*). Nous pouvons également voir les états intermédiaires dans la mise en œuvre des requêtes de synchronisation (procédure d'autorisation et éventuellement procédure d'annulation ou de refus).

Considérons par exemple le cas où *i* est asynchrone. Deux situations peuvent se présenter : (i). *i* se synchronise à un utilisateur *j* au moyen d'une requête « *j ! I_Follow_You* » ou, (ii). *i* reçoit une invitation de synchronisation en provenance d'un utilisateur *j* au moyen d'une requête « *j ? You_Follow_Me* ». Dans les deux cas, pour créer ces relations de synchronisation, il est nécessaire que la condition *root(j, i)* définie précédemment ne soit pas satisfaite, car sinon cela conduirait à l'établissement de boucles dans une structure (le SDT) qui est nécessairement un arbre.

Une fois qu'une requête de synchronisation a été effectuée, l'étape suivante consiste à rester dans un état d'attente (d'une autorisation ou d'un refus, voire d'une éventuelle annulation de la requête de synchronisation) avant d'aller dans l'état correspondant soit au succès (état *sync(i, j)*) soit à l'échec (état *async(i)*) de la requête de synchronisation.

Le traitement d'un requête (« I_Leave_You » permet de repasser de l'état synchronisé *sync(i, j)* à l'état asynchrone *async(i)*.

Dans la *Figure 42*, nous pouvons observer le comportement de l'utilisateur *i* avec lequel un autre utilisateur *k* désire se synchroniser. Cet automate ressemble à l'automate précédent et doit pouvoir être analysé par le lecteur sans trop de difficultés.

En fusionnant par places communes les automates étendus présentés dans la *Figure 41* et la *Figure 42*, nous obtenons la description de l'ensemble des comportements possibles du point de vue de l'utilisateur *i*.

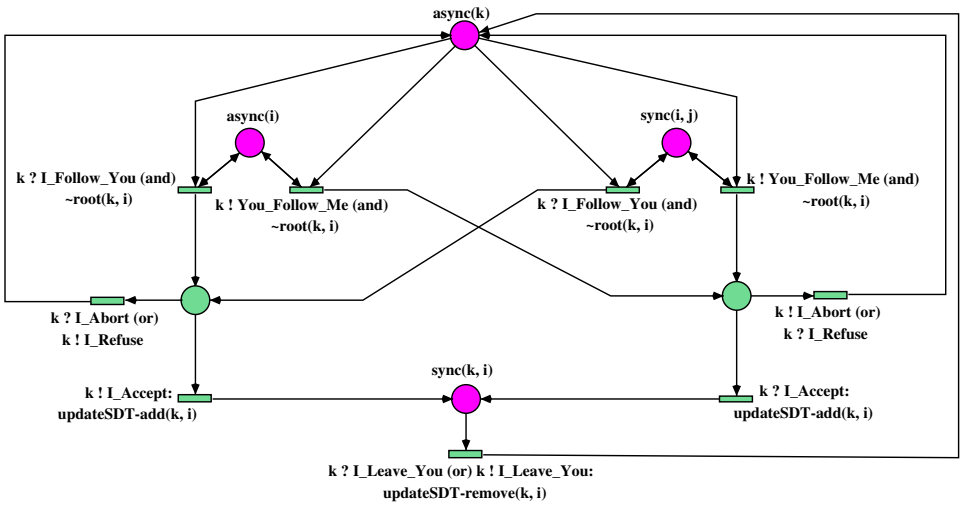


Figure 42. Synchronisation d'un utilisateur k avec l'utilisateur i (modèle de synchronisation de base)

III.3.2. Formalisation par des réseaux de Petri

Partant de la formalisation effectuée dans le paragraphe précédent, notre objectif est maintenant, dans un but de vérification formelle, d'effectuer une modélisation globale d'un ensemble d'utilisateurs qui effectuent au sein d'une session de navigation coopérative tout un ensemble d'actions de synchronisation.

Notre choix s'est porté sur les réseaux de Petri, du fait de la disponibilité de l'outil d'analyse TINA [Berthomieu-04] [Tina] développé au LAAS-CNRS. Une motivation derrière le choix de TINA a été en particulier l'existence dans cet outil de mécanismes de maîtrise de l'explosion combinatoire, sachant que nous appréhendons un nombre important d'états pour une analyse exhaustive des mécanismes de synchronisation que nous avons proposés, dès que le nombre d'utilisateurs connectés à une session de navigation coopérative va croître. Cette formalisation en réseaux de Petri sera présentée de la manière suivante :

- dans un premier temps, nous montrons comment nous représentons en RdP l'ensemble des SDTs d'une session de navigation coopérative ;
- dans un deuxième temps, nous modélisons les différentes actions de synchronisation permettant de créer et supprimer des relations de synchronisation, et comment elles modifient les SDTs.

Le modèle du réseau de Petri global est engendré, comme nous le verrons, au moyen de scripts par composition de modèles élémentaires et génériques. Ce modèle global servira de base à l'analyse effectuée dans le paragraphe III.4.

III.3.2.1. Modélisation des SDTs

Soit N le nombre d'utilisateurs qui se sont connectés à une session de navigation coopérative, avec i, j, k $\in$ [1, N]. Nous faisons l'hypothèse que les notations F_{ij}, *Fij* et *F_i_j*, $\forall$i,j sont équivalentes.

Pour modéliser un ensemble de SDTs présents dans une session de navigation coopérative, nous définissons plusieurs ensembles de places, qui ont la signification suivante :

- $F_{i,j}$ $\forall i, j \in [1, N], i \neq j$: Cette place est marquée si le nœud *i* est fils du nœud *j* (ce qui exprime que l'utilisateur *i* est synchronisé à l'utilisateur *j*) ;

- $NF_{i,j}$ $\forall i, j \in [1, N]$, $i \neq j$: Cette place est la place complémentaire de $F_{i,j}$ (cette place marquée indique donc que l'utilisateur *i* n'est pas synchronisé à l'utilisateur *j*) ;
- $R_{i,j}$ $\forall i, j \in [1, N]$: Cette place est marquée si le nœud *i* appartient à un SDT dont la racine est le nœud *j* ; si un nœud *i* est racine d'un SDT, alors par définition la place $R_{i,i}$ est marquée ;
- $NR_{i,j}$ $\forall i, j \in [1, N]$: Cette place est la place complémentaire de $R_{i,j}$.

Tout marquage cohérent de cet ensemble de places caractérise un ensemble de SDTs, et donc un état des relations de synchronisation existant entre les différents utilisateurs d'une même session de navigation coopérative. Au delà des conditions triviales liées à l'existence de places complémentaires, les propositions suivantes doivent être vérifiées pour assurer cette cohérence :

- $F_{i,j} \rightarrow \neg\exists(F_{i,k})$, avec $j \neq k$
- $R_{i,j} \rightarrow \neg\exists(R_{i,k})$, avec $j \neq k$
- $F_{i,j} \rightarrow \neg\exists(R_{i,i})$, avec $i \neq j$
- $F_{i,j} \wedge R_{j,k} \rightarrow \exists R_{i,k}$

À partir des définitions précédentes, nous constatons que l'état de synchronisation d'un utilisateur *i* se caractérise essentiellement par le marquage de deux places :

- Si la place *Fij* est marquée, alors l'utilisateur *i* est synchronisé à l'utilisateur *j*
- Si la place *Rik* est marquée, alors la navigation de l'utilisateur *i* suit celle de l'utilisateur *k*, qui est nécessaire asynchrone (racine du SDT).

Un exemple typique d'un état de synchronisation d'une session de navigation coopérative avec 5 utilisateurs est présenté dans la *Figure 43*. Ici nous pouvons observer qu'il existe déjà un certain nombre de relations de synchronisation (flèches noires) : l'utilisateur *2* est synchronisé à l'utilisateur *1* (F_2_1), et les utilisateurs *4* et *5* sont synchronisés à l'utilisateur *3* (F_4_3 et F_5_3). Nous pouvons également voir la définition de la racine du SDT pour chaque utilisateur (flèche pointillée) : la racine des utilisateurs *1* et *2* est l'utilisateur *1*, et la racine des utilisateurs *3*, *4* et *5* est l'utilisateur *3*.

Le nombre de places nécessaires pour représenter l'ensemble des SDTs dépend directement du nombre *N* d'utilisateurs. Nous pouvons remarquer que nous avons besoin de $4N^2-2N$ places dont $2N^2-N$ sont marquées pour décrire un état de synchronisation quelconque. Les places complémentaires ont été introduites pour réaliser des tests à zéro, ce qui est possible vu le que le nombre d'utilisateurs est fini par définition au niveau de notre modélisation.

Le réseau de Petri modélisant l'ensemble des SDTs peut devenir rapidement très complexe en fonction du nombre d'utilisateurs. Pour cette raison, nous avons défini des composants de base que nous composons au moyen de scripts, car il est bien sûr hors de question d'éditer le réseau de Petri place par place et transition par transition. Ces scripts ont été créés en utilisant le langage Tcl [TCL].

Nous allons maintenant aborder comment modéliser en réseau de Petri les mécanismes de synchronisation conduisant à la création et à la suppression de relations de synchronisation entre utilisateurs d'une même session de navigation coopérative.

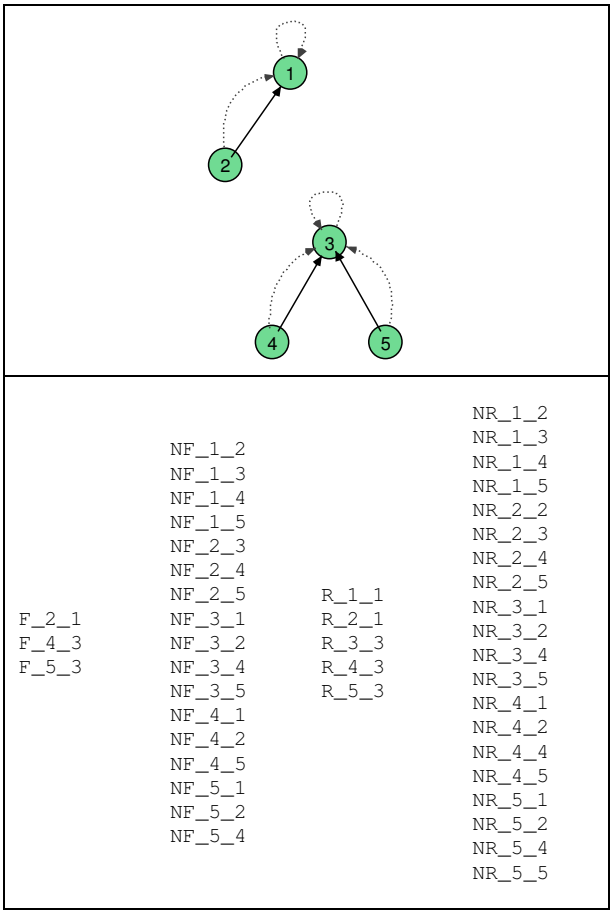


Figure 43. Exemple de description d'un état de synchronisation

III.3.2.2. Modélisation de la création de relations de synchronisation

Dans notre modèle de synchronisation, des relations de synchronisation peuvent être établies au moyen de la requête « Follow » qui peut se décliner, comme nous l'avons vu précédemment, sous deux formes syntaxiques : « I_Follow_You » ou « You_Follow_Me ». Nous allons modéliser cette requête de synchronisation sous la forme « Follow_i_j » qui signifie que l'utilisateur *i* se synchronise à l'utilisateur *j*, ce qui peut être la conséquence soit d'une requête « I_Follow_You(j) » (effectuée par l'utilisateur *i*), soit d'une requête « You_Follow_Me(i) » (effectuée par l'utilisateur *j*).

Le composant « Follow » qui traite cette action de synchronisation est présenté dans la *Figure 44*. Ce composant a une structure de réseau de Petri mais utilise des conventions de notation pour garder une représentation compacte du réseau indépendante du nombre d'utilisateurs présents dans la session de navigation coopérative. Ces notations sont les suivantes :

- *k* /*k* <> *i* signifie pour un *k* tel que *k* différent de *i*
- **l* /*l* <> *i* signifie pour tout *l* tel que *l* différent de *i*

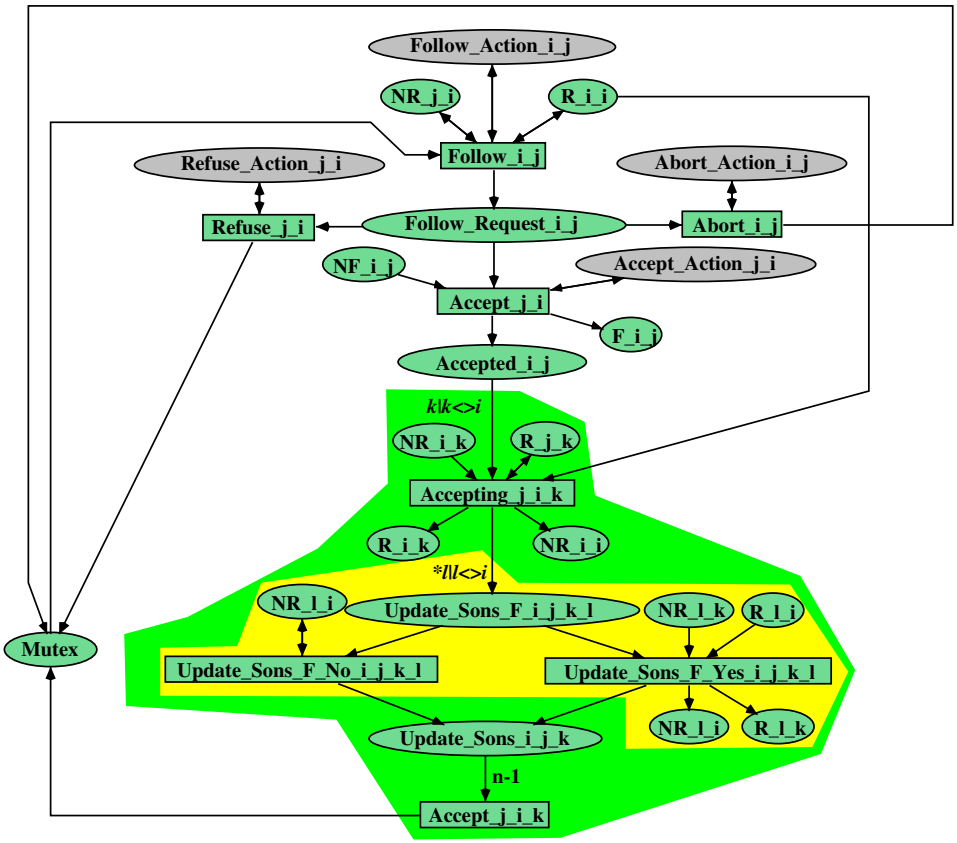


Figure 44. Modélisation du composant « Follow »

Les places « Follow_Action_i_j », « Refuse_Action_j_i », « Abort_Action_i_j » et « Accept_Action_j_i » sont des places introduites pour limiter le non déterminisme du réseau global et ainsi contrôler l'explosion combinatoire du graphe de marquage associé au réseau. Ainsi, par exemple, si la place « Refuse_Action_j_i » est marquée dans le marquage initial du réseau, la transition « Refuse_j_i » pourra être sensibilisée, et ne le sera jamais dans le cas contraire. Ces différentes places sont initialisées selon le type d'analyse désiré et nous permettent de manière très simple de contrôler l'occurrence des différentes actions que nous souhaitons analyser dans une spécification.

Le composant « Follow » de la *Figure 44* peut s'expliquer assez simplement vis à vis des trois étapes qui doivent être traitées lors de l'exécution d'une requête de synchronisation « Follow_i_j », comme nous allons l'analyser ci-dessous :

- La première étape consiste à vérifier que les conditions pour que « Follow_i_j » puisse s'exécuter sont satisfaites, à savoir que :
  - l'utilisateur que va se synchroniser (i.e. l'utilisateur *i*) est asynchrone (ce qui signifie qu'il n'est pas déjà synchronisé à un autre utilisateur)
  - cet utilisateur (i.e. toujours *i*) n'est pas la racine du SDT auquel appartient l'utilisateur avec qui *i* désire se synchroniser (à savoir *j*).
- La deuxième étape, localisée à l'intérieur de l'aire colorée externe, consiste à chercher l'utilisateur qui est, dans l'état courant de la session, racine du SDT auquel appartient l'utilisateur avec qui *i* désire se synchroniser.
- La troisième étape, localisée à l'intérieur de l'aire colorée interne, consiste à mettre à jour pour tout utilisateur la racine de son SDT. Ainsi, tout utilisateur qui, avant l'exécution de la requête « Follow_i_j », avait comme racine l'utilisateur *i*, aura comme racine, après exécution de la requête, la racine du SDT auquel appartient l'utilisateur *j*.

Le réseau global associé au composant « Follow » dépend du nombre d'utilisateurs présents dans une session de navigation coopérative. Il est engendré par le script présenté dans la *Figure 45*.

```
# User "i" makes a "I_Follow_You" on user "j"
# User "j" makes a "You_Follow_Me" on user "i"
for {set i 1} {$i <= $n} {incr i} {
  for {set j 1} {$j <= $n} {incr j} {
    if {!($i == $j)} {
      puts "tr Follow_${i}_${j} R_${i}_${i} NR_${j}_${i}
        Follow_Action_${i}_${j} Mutex ->
        Follow_Action_${i}_${j} Follow_Request_${i}_${j} R_${i}_${i}
        NR_${j}_${i} "
      puts "tr Refuse_${j}_${i} Follow_Request_${i}_${j}
        Refuse_Action_${j}_${i} ->
        Refuse_Action_${j}_${i} Mutex"
      puts "tr Abort_${i}_${j} Follow_Request_${i}_${j}
        Abort_Action_${i}_${j} ->
        Abort_Action_${i}_${j} Mutex"
      puts "tr Accept_${j}_${i} Follow_Request_${i}_${j}
        Accept_Action_${j}_${i} NF_${i}_${j} ->
        Accept_Action_${j}_${i} Accepted_${i}_${j} F_${i}_${j}"
      for {set k 1} {$k <= $n} {incr k} {
        if {!($k == $i)} {
          puts "tr Accepting_${j}_${i}_${k} Accepted_${i}_${j}
            R_${j}_${k} R_${i}_${i} NR_${i}_${k} ->
            NR_${i}_${i} R_${i}_${k} R_${j}_${k}"
          for {set l 1} {$l <= $n} {incr l} {
            if {!($l == $i)} {
              puts "tr Accepting_${j}_${i}_${k} ->
                Update_Sons_F_${i}_${j}_${k}_${l}"
              puts "tr Update_Sons_F_No_${i}_${j}_${k}_${l}
                Update_Sons_F_${i}_${j}_${k}_${l} NR_${l}_${i} ->
                NR_${l}_${i} Update_Sons_${i}_${j}_${k}"
              puts "tr Update_Sons_F_Yes_${i}_${j}_${k}_${l}
                Update_Sons_F_${i}_${j}_${k}_${l} R_${l}_${i}
                NR_${l}_${k} ->
                R_${l}_${k} NR_${l}_${i} Update_Sons_${i}_${j}_${k}"
            } # if
          } # for l
          puts "tr Accept_${j}_${i}_${k}
            Update_Sons_${i}_${j}_${k}*[expr $n - 1] ->
            Mutex"
        } # if
      } # for k
    } # if
  } # for j
} # for i
```

Figure 45. Script Tcl de génération du composant « Follow »

A titre d'exemple, nous présentons, dans la *Figure 46*, le réseau de Petri engendré pour une session de navigation coopérative réduite à deux utilisateurs (au delà de deux utilisateurs, le réseau de Petri devient illisible).

Dans le but d'illustrer l'exécution d'une action de synchronisation « Follow_i_j » nous présentons, dans la *Figure 47*, un exemple avec quatre utilisateurs présents dans une session de navigation coopérative. L'intuition de l'action de synchronisation « Follow_2_1 » est évidente sur les arbres de synchronisation (SDTs) et elle est détaillée pour les places représentant les arbres de synchronisation sous forme de réseaux de Petri. Les places modifiées lors de l'exécution de « Follow_2_1 » sont marquées en gras.

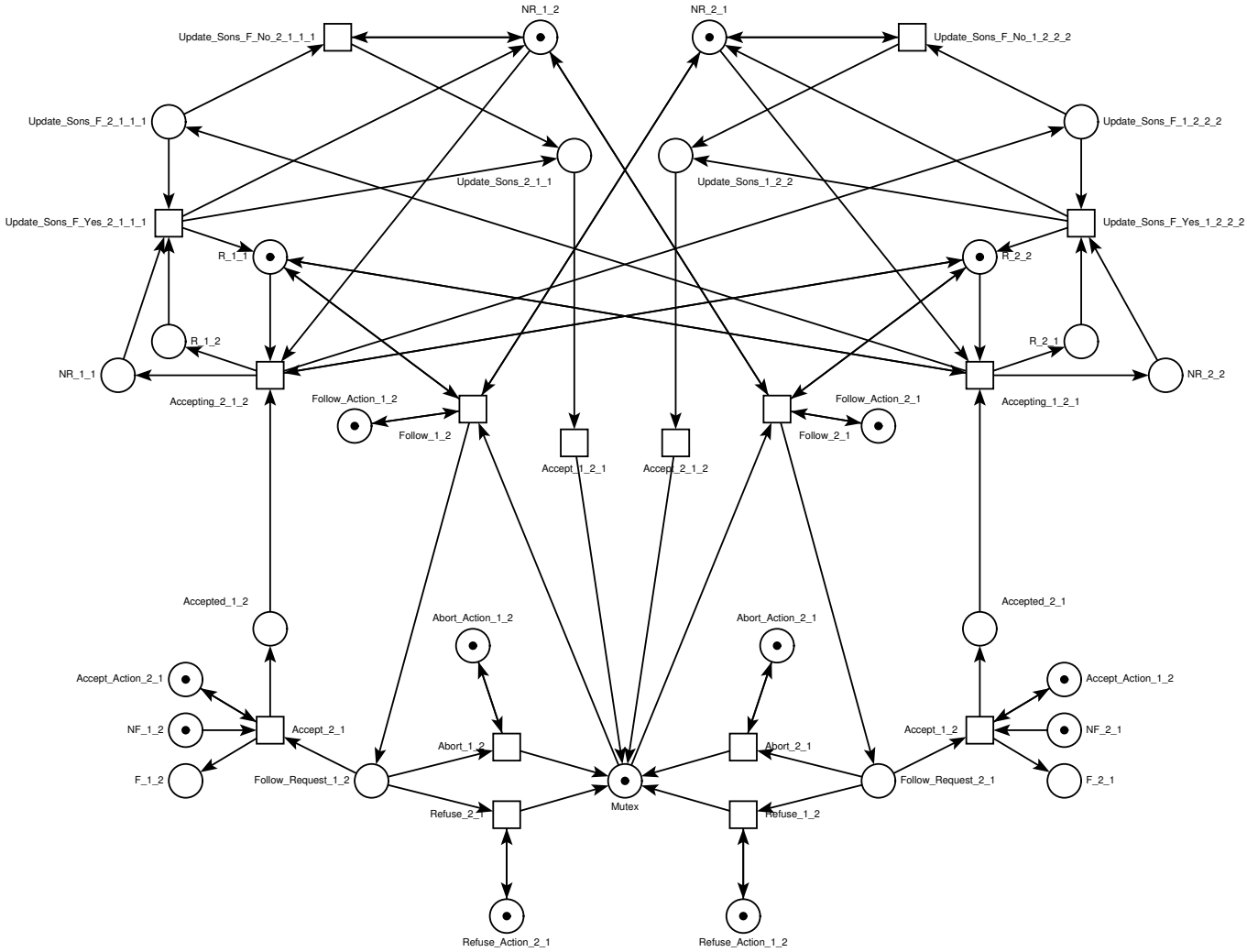


Figure 46. Réseau de Petri engendré pour le composant « Follow » avec deux utilisateurs

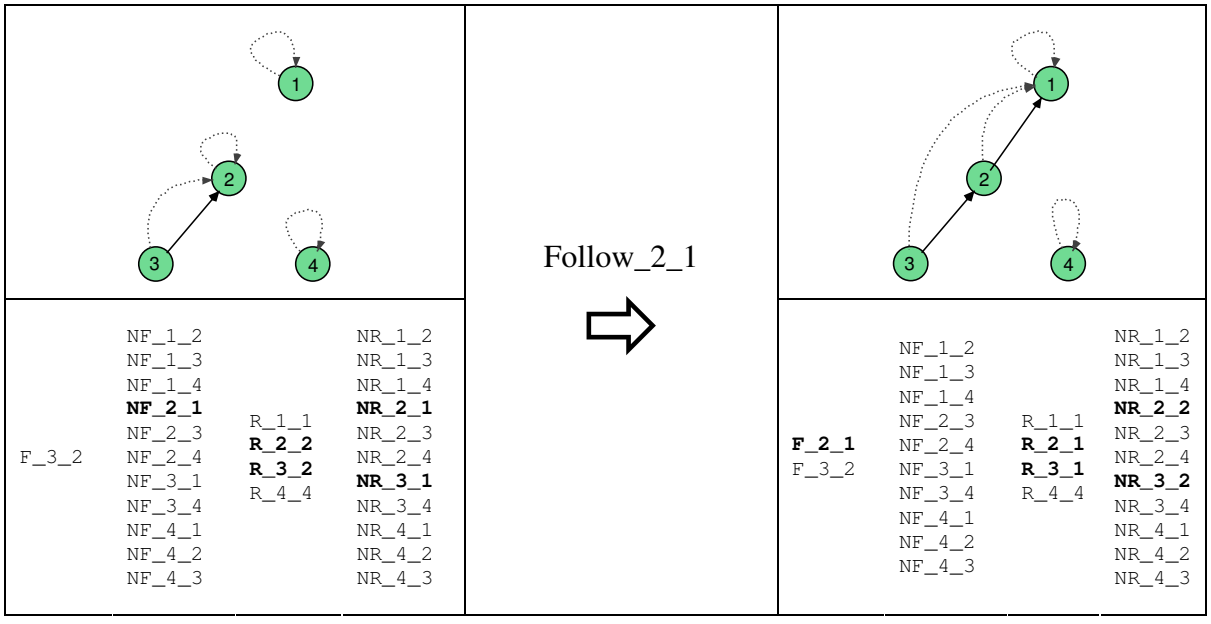


Figure 47. Illustration de l'exécution de l'action « Follow_2_1 »

III.3.2.3. Modélisation de la suppression de relations de synchronisation

Dans notre modèle de synchronisation, des relations de synchronisation peuvent être supprimées au moyen de la requête « Leave » qui se décline, comme nous l'avons vu précédemment, sous la forme syntaxique : « I_Leave_You » qui peut être à l'initiative de l'une ou l'autre des extrémités de la relation de synchronisation. Nous allons modéliser cette requête de synchronisation sous la forme « Leave_i_j » qui signifie que la relation de synchronisation établie entre les utilisateurs *i* et *j* (*i* est synchronisé à *j*) doit être supprimée.

Le composant « Leave » qui traite cette action de synchronisation est présenté dans la *Figure 48*, qui utilise les mêmes conventions de notation que le composant « Follow » de la *Figure 44*.

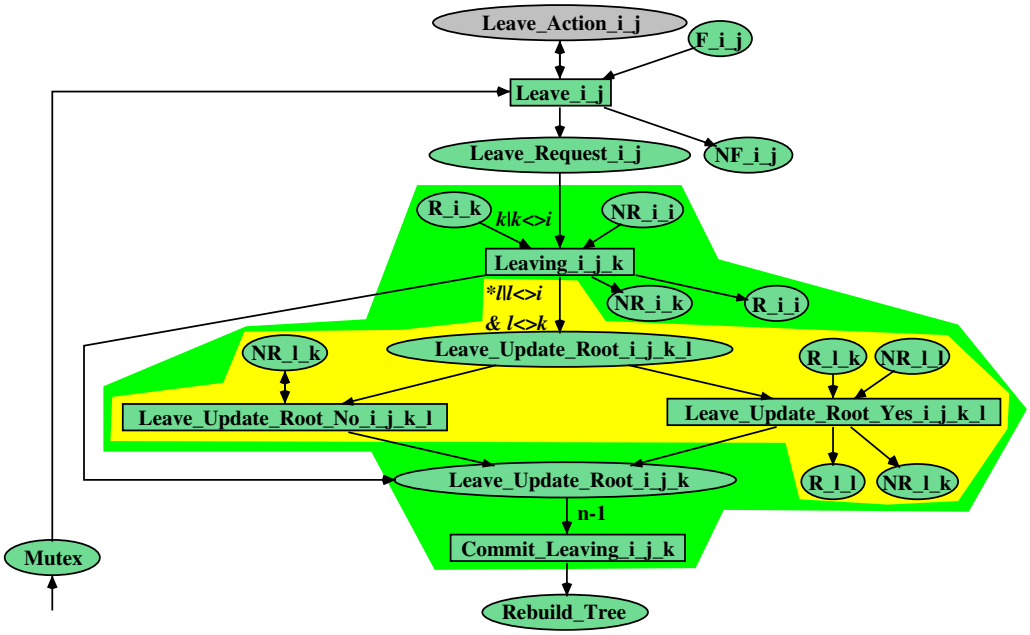


Figure 48. Modélisation du composant « Leave »

La modélisation du mécanisme de suppression d'une relation de synchronisation est nettement plus complexe que celle de la création d'une relation de synchronisation. Elle est constituée de deux étapes :

- La première consiste à supprimer la relation de synchronisation, ce qui, dans le cas général, rend la représentation des SDTs incohérente (étape correspondant au composant « Leave » dont la modélisation est présentée dans la *Figure 48*).
- La deuxième consiste à reconstruire la représentation des SDTs pour la rendre cohérente (étape correspondante au composant « Rebuild_Tree » dont la modélisation est présentée dans la *Figure 52*).

La place « Leave_Action_i_j » joue le même rôle que les places « Follow_Action_i_j », « Refuse_Action_j_i », « Abort_Action_i_j » et « Accept_Action_j_i » définies pour le composant « Follow ». Ainsi, si la place « Leave_Action_i_j » est marquée, la transition « Leave_i_j » pourra être sensibilisée, et ne le sera jamais dans le cas contraire. Cette place est initialisée selon le type d'analyse désiré.

Le composant « Leave » de la *Figure 48* peut s'expliquer assez simplement vis à vis des trois étapes qui doivent être traitées lors de l'exécution d'une requête de synchronisation « Leave_i_j », comme nous allons l'analyser ci-dessous :

- la première étape consiste à vérifier qu'il existe effectivement existe une relation de synchronisation entre les utilisateurs *i* et *j*. Si c'est le cas, alors la relation de synchronisation entre *i* et *j* est immédiatement supprimée ;
- la deuxième étape, localisée dans la *Figure 48* à l'intérieur de l'aire colorée externe, consiste à chercher le nœud racine du SDT auquel appartenait les utilisateurs *i* et *j* (avant que la relation de synchronisation ne soit supprimée) ;
- la troisième étape, localisée dans la *Figure 48* à l'intérieur de l'aire colorée interne, consiste, pour tout utilisateur ayant comme nœud racine le nœud identifié dans l'étape précédente, à faire qu'il se considère temporairement comme son propre nœud racine. Cette approche crée bien sûr des incohérences au niveau de la représentation des SDTs qui seront corrigées ultérieurement grâce au composant « Rebuild_Tree ».

Le script pour engendrer un réseau de Petri à partir du composant « Leave » est présenté dans la *Figure 49*.

```
# User "i" makes a "I_Leave_You" on user "j"
# User "j" makes a "I_Leave_You" on user "i"
for {set i 1} {$i <= $n} {incr i} {
  for {set j 1} {$j <= $n} {incr j} {
    if {!($i == $j)} {
      puts "tr Leave_${i}_${j} F_${i}_${j} Leave_Action_${i}_${j}
        Mutex ->
        Leave_Action_${i}_${j} Leave_Request_${i}_${j} NF_${i}_${j}"
      for {set k 1} {$k <= $n} {incr k} {
        if {!($k == $i)} {
          puts "tr Leaving_${i}_${j}_${k} Leave_Request_${i}_${j}
            R_${i}_${k} NR_${i}_${i} ->
            NR_${i}_${k} R_${i}_${i} Leave_Update_Root_${i}_${j}_${k}"
            for {set l 1} {$l <= $n} {incr l} {
              if {!($l == $i) && !($l == $k)} {
                puts "tr Leaving_${i}_${j}_${k} ->
                  Leave_Update_Root_${i}_${j}_${k}_${l}"
                puts "tr Leave_Update_Root_No_${i}_${j}_${k}_${l}
                  Leave_Update_Root_${i}_${j}_${k}_${l} NR_${l}_${k} ->
                  NR_${l}_${k} Leave_Update_Root_${i}_${j}_${k}"
                puts "tr Leave_Update_Root_Yes_${i}_${j}_${k}_${l}
                  Leave_Update_Root_${i}_${j}_${k}_${l} R_${l}_${k}
                  NR_${l}_${l} ->
                  NR_${l}_${k} R_${l}_${l}
                  Leave_Update_Root_${i}_${j}_${k}"
              } if
            } # for l
            puts "tr Commit_Leaving_${i}_${j}_${k}
              Leave_Update_Root_${i}_${j}_${k}*[expr $n - 1] ->
              Rebuild_Tree"
        } # if
      } # for k
    } # if
  } # for j
} # for i
```

Figure 49. Script Tcl de génération du composant « Leave »

Toujours dans le cas d'une session de navigation coopérative réduite à deux utilisateurs, le réseau de Petri associé au composant « Leave » est présenté dans la *Figure 50*.

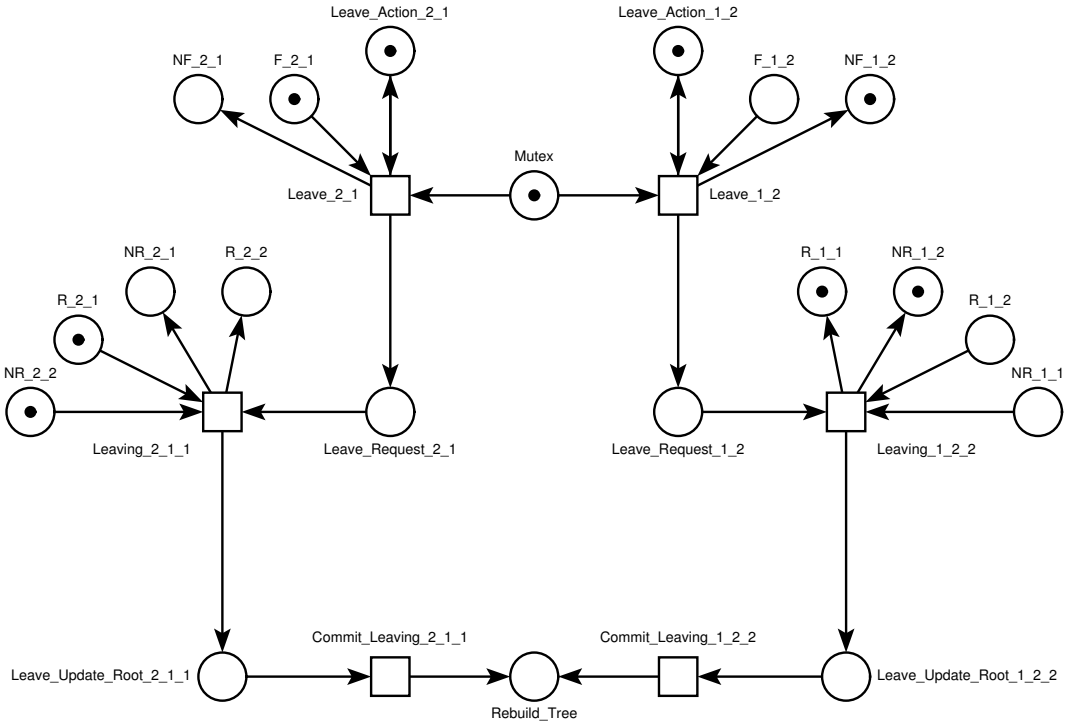


Figure 50. Réseau de Petri engendré pour le composant « Leave » avec deux utilisateurs.

Pour illustrer l'exécution d'une action de synchronisation « Leave_i_j » nous présentons, dans la *Figure 51* un exemple avec quatre utilisateurs. L'intuition de l'action de synchronisation « Leave_2_1 » est évidente sur les arbres de synchronisation (SDTs) et elle est détaillée pour les places représentant les arbres de synchronisation sous forme de réseaux de Petri. Les places modifiées lors de l'exécution de « Leave_2_1 » sont marquées en gras.

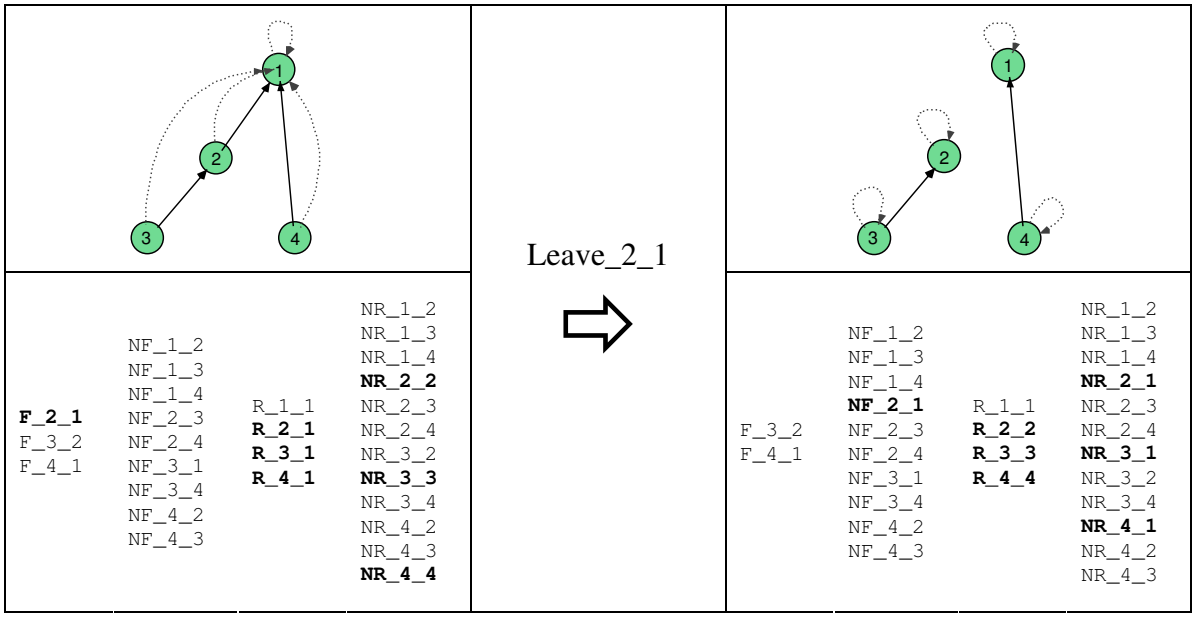


Figure 51. Illustration de l'exécution de l'action « Leave_2_1 » (composant « Leave »)

Comme nous l'avons souligné auparavant, la modélisation du mécanisme de suppression d'une relation de synchronisation est effectuée en deux étapes. La première étape (réalisée par le composant « Leave »), destinée à supprimer une relation de synchronisation, rend la représentation des SDTs incohérente. Le composant « Rebuild_Tree » a pour vocation de rendre cette représentation des SDTs de nouveau cohérente. Ce composant est présenté dans la *Figure 52* et utilise les mêmes conventions de notation que les composants « Follow » et « Leave ».

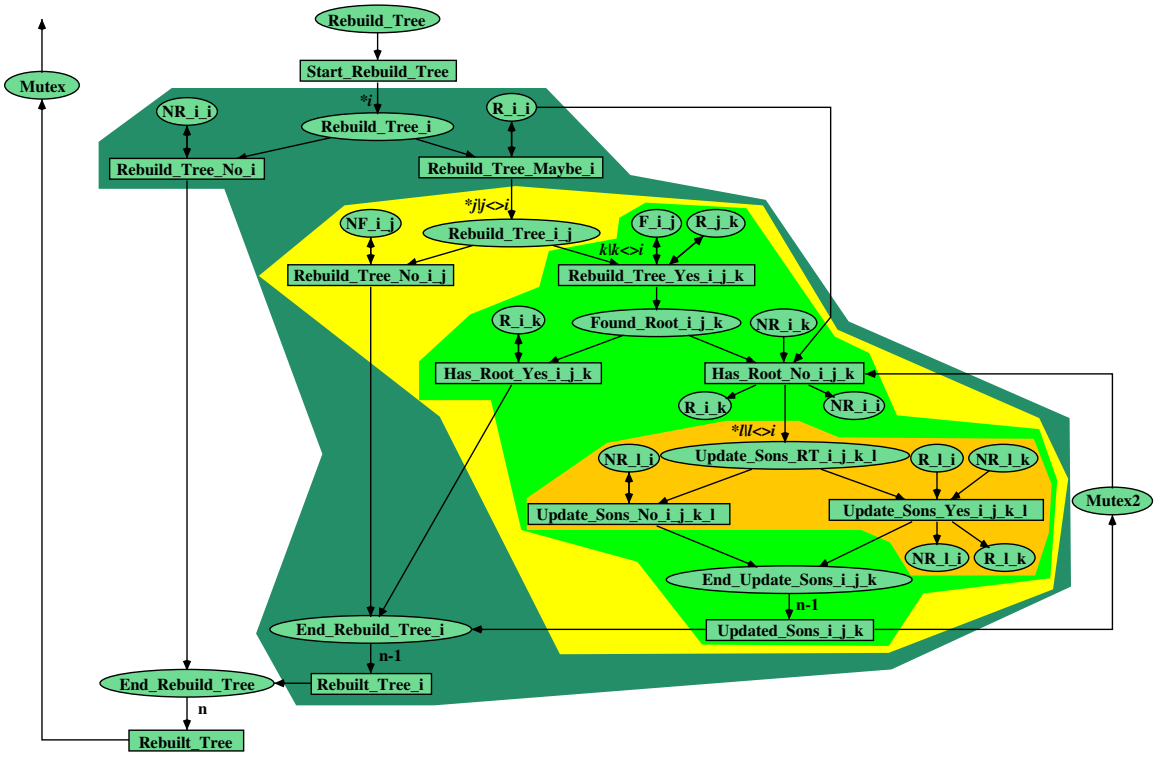


Figure 52. Modélisation du composant « Rebuild_Tree »

Comme nous pouvons le voir ci-dessus, la modélisation du composant « Rebuild_Tree » est assez complexe, et demande, pour effectuer les différentes mises à jour des relations racine de chaque nœud, de parcourir plusieurs fois l'ensemble des nœuds.

Le script pour engendrer un réseau de Petri à partir du composant « Rebuild_Tree » est présenté dans la *Figure 53*.

```
# SDT rebuilding after a Leave action
puts "tr Start_Rebuild_Tree Rebuild_Tree ->"
for {set i 1} {$i <= $n} {incr i} {
  puts "tr Start_Rebuild_Tree -> Rebuild_Tree_${i}"
  puts "tr Rebuild_Tree_No_${i} Rebuild_Tree_${i} NR_${i}_${i} ->
    NR_${i}_${i} End_Rebuild_Tree"
  puts "tr Rebuild_Tree_Maybe_${i} Rebuild_Tree_${i} R_${i}_${i} ->
    R_${i}_${i}"
  for {set j 1} {$j <= $n} {incr j} {
    if {!($i == $j)} {
      puts "tr Rebuild_Tree_Maybe_${i} -> Rebuild_Tree_${i}_${j}"
      puts "tr Rebuild_Tree_No_${i}_${j} Rebuild_Tree_${i}_${j}
        NF_${i}_${j} ->
        NF_${i}_${j} End_Rebuild_Tree_${i}"
      for {set k 1} {$k <= $n} {incr k} {
        if {!($k == $i)} {
          puts "tr Rebuild_Tree_Yes_${i}_${j}_${k} Rebuild_Tree_${i}_${j}
            F_${i}_${j} R_${j}_${k} ->
            F_${i}_${j} R_${j}_${k} Found_Root_${i}_${j}_${k}"
          puts "tr Has_Root_Yes_${i}_${j}_${k} R_${i}_${k}
            Found_Root_${i}_${j}_${k} ->
            R_${i}_${k} End_Rebuild_Tree_${i}"
          puts "tr Has_Root_No_${i}_${j}_${k} Found_Root_${i}_${j}_${k}
            NR_${i}_${k} R_${i}_${i} Mutex2 ->
            R_${i}_${k} NR_${i}_${i}"
          for {set l 1} {$l <= $n} {incr l} {
            if {!($l == $i)} {
              puts "tr Has_Root_No_${i}_${j}_${k} ->
                Update_Sons_RT_${i}_${j}_${k}_${l}"
              puts "tr Update_Sons_RT_No_${i}_${j}_${k}_${l}
                Update_Sons_RT_${i}_${j}_${k}_${l} NR_${l}_${i} ->
                NR_${l}_${i} End_Update_Sons_${i}_${j}_${k}"
              puts "tr Update_Sons_RT_Yes_${i}_${j}_${k}_${l}
                Update_Sons_RT_${i}_${j}_${k}_${l} R_${l}_${i}
                NR_${l}_${k} ->
                NR_${l}_${i} R_${l}_${k}
                End_Update_Sons_${i}_${j}_${k}"
            } # if
          } # for l
          puts "tr Updated_Sons_${i}_${j}_${k}
            End_Update_Sons_${i}_${j}_${k}*[expr $n - 1] ->
            End_Rebuild_Tree_${i} Mutex2"
        } # if
      } # for k
    } # if
  } # for j
  puts "tr Rebuilt_Tree_${i} End_Rebuild_Tree_${i}*[expr $n - 1] ->
    End_Rebuild_Tree"
} # for i
puts "tr Rebuilt_Tree End_Rebuild_Tree*[expr $n] -> Mutex"
```

Figure 53. Script Tcl de génération du composant « Rebuild_Tree »

Dans le cas d'une session de navigation coopérative réduite à deux utilisateurs, le réseau de Petri associé au composant « Rebuild_Tree » est présenté dans la *Figure 54*.

En reprenant l'exemple de la *Figure 51* nous pouvons analyser en détail le rôle du composant « Rebuild_Tree » pour rendre cohérente la représentation des SDTs suite à une action de synchronisation « Leave ». Ceci est présenté dans la *Figure 55* ci-dessous. Comme dans les exemples précédents les places modifiées lors de l'exécution de « Rebuild_Tree » sont marquées en gras.

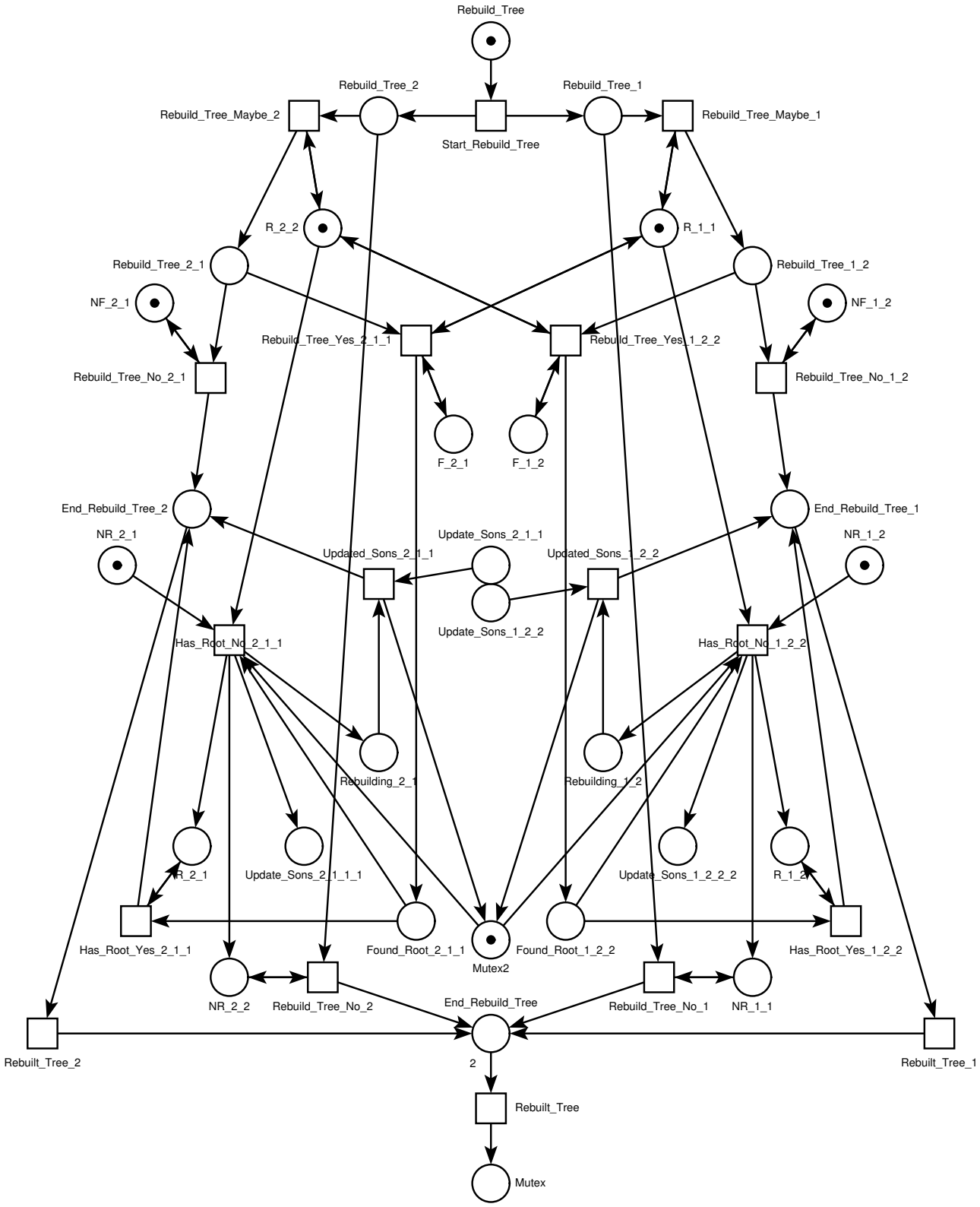


Figure 54. Réseau de Petri engendré par le composant « Rebuild_Tree » avec deux utilisateurs

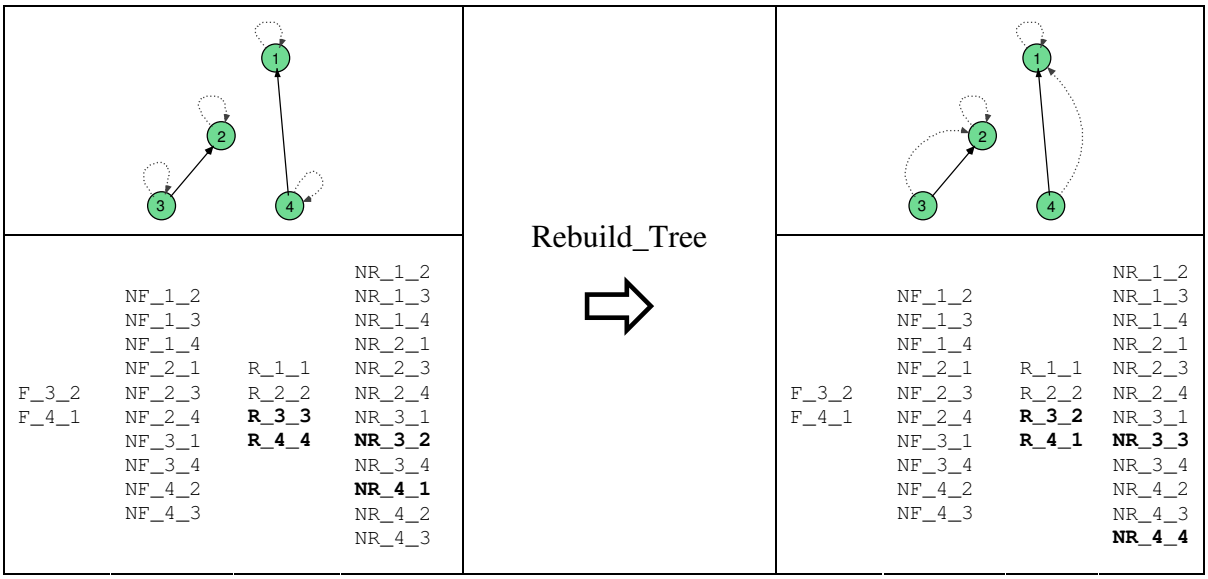


Figure 55. Rôle du composant « Rebuild_Tree »

III.3.2.4. Modélisation du processus de navigation

La dernière action que nous considérons dans notre modèle de synchronisation est l'action « Browse » qui se décline sous deux formes : « Browse_j » et « Browse_i_j ». « Browse_j » représente une action de navigation réalisée par un utilisateur *j* qui est asynchrone (i.e. *j* est la racine d'un SDT et a donc la possibilité de naviguer de sa propre initiative) alors que « Browse_i_j » représente l'action de navigation réalisée par *i* suite à une action de navigation réalisée par *j* (i.e. *i* ne peut pas naviguer de sa propre initiative, sa navigation est synchronisée à celle de *j*).

Le composant « Browse », qui traite la synchronisation des actions de navigation, est présenté dans la *Figure 56* qui utilise les mêmes conventions de notation que les composants précédents.

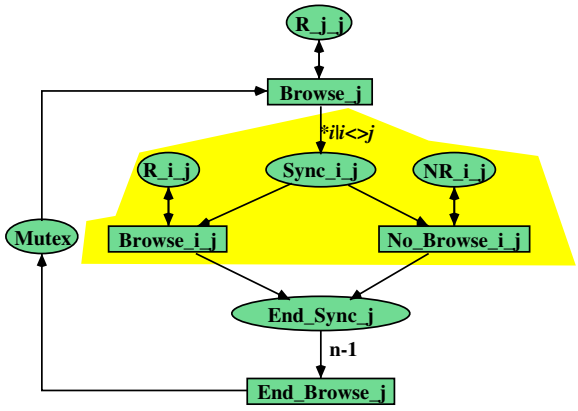


Figure 56. Modélisation du composant « Browse »

Le composant « Browse » est beaucoup plus simple que les autres composants car il n'est pas nécessaire ici de modifier la structure des SDTs, mais uniquement de les parcourir pour propager l'action de navigation réalisée au niveau de la racine d'un SDT sur l'ensemble des nœuds de ce SDT. Nous voyons ainsi dans le composant « Browse » que :

- l'action de navigation « Browse_j » ne peut se réaliser que si l'utilisateur *j* est asynchrone (*j* est la racine d'un SDT, donc la place *R_j_j* est marquée) ;

- une fois que l'action « Browse_j » est réalisée, il suffit de parcourir le SDT de racine *j* et de réaliser l'action « Browse_i_j » pour tout utilisateur *i* appartenant au SDT de racine *j*.

Le script pour engendrer un réseau de Petri à partir du composant « Browse » est présenté dans la *Figure 57*.

```
# User "j" executes a browsing action
for {set j 1} {$j <= $n} {incr j} {
  puts "tr Browse_${j} R_${j}_${j} Mutex -> R_${j}_${j}"
  for {set i 1} {$i <= $n} {incr i} {
    if {!($j == $i)} {
      puts "tr Browse_${j} -> Sync_${i}_${j}"
      puts "tr Browse_${i}_${j} R_${i}_${j} Sync_${i}_${j} ->
        R_${i}_${j} End_Sync_${j}"
      puts "tr No_Browse_${i}_${j} NR_${i}_${j} Sync_${i}_${j} ->
        NR_${i}_${j} End_Sync_${j}"
    } # if
  } # for i
  puts "tr End_Browse_${j} End_Sync_${j}*[expr $n - 1] -> Mutex"
} # for j
```

Figure 57. Script Tcl de génération du composant « Browse »

De nouveau, dans le cas d'une session de navigation coopérative réduite à deux utilisateurs, le réseau de Petri associé au composant « Browse » est présenté dans la *Figure 58*.

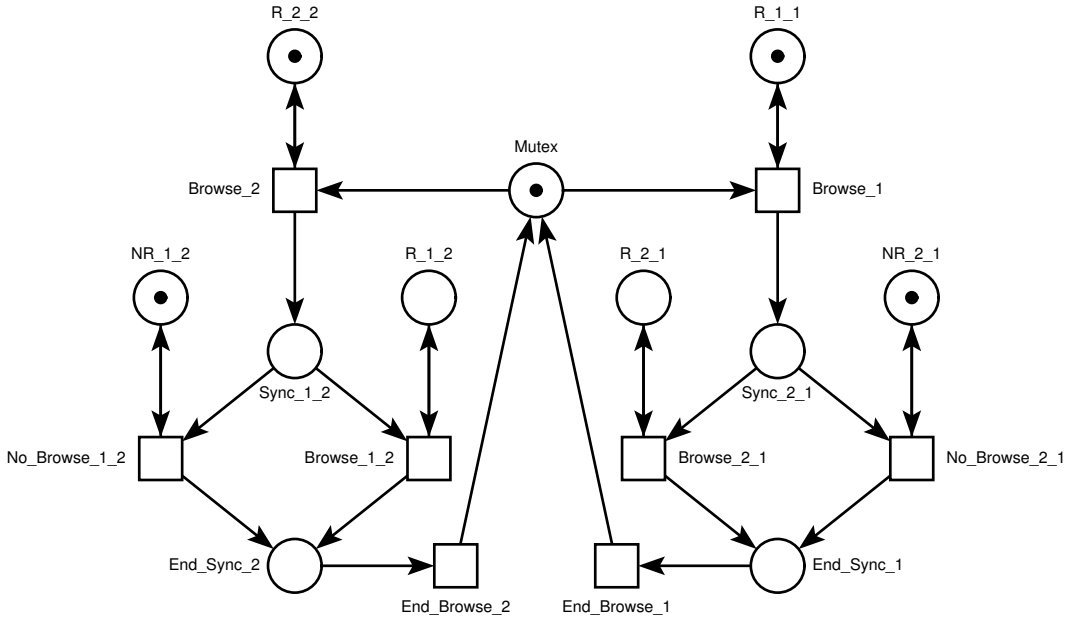


Figure 58. Réseau de Petri engendré par le composant « Browse » avec deux utilisateurs

III.3.2.5. Tout ensemble

Dans ce paragraphe, nous combinons les composants détaillés précédemment dans le but d'obtenir un réseau de Petri formalisant globalement notre modèle de synchronisation de base.

Le script pour engendrer ce réseau de Petri à partir des composants « Follow », « Leave », « Rebuild_Tree » et « Browse » est présenté dans la *Figure 59*, avec comme

marquage initial une configuration des places où tous les utilisateurs sont asynchrones (aucune relation de synchronisation n'a été établie entre les différents utilisateurs).

```
#!/bin/sh
#\
exec tclsh "$0" -- "$@"

proc main {} {
  global argv

  set liste ""
  if {"" == [set n [lindex $argv 1]]} {set n 2}
    set n [string trimleft $n 0]

# Initial Follow_Action
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl Follow_Action_${i}_${j} (1)"
      } # if
    } # for j
  } # for i

# Initial Accept_Action
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl Accept_Action_${i}_${j} (1)"
      } # if
    } # for j
  } # for i

# Initial Refuse_Action
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl Refuse_Action_${i}_${j} (1)"
      } # if
    } # for j
  } # for i

# Initial Abort_Action
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl Abort_Action_${i}_${j} (1)"
      } # if
    } # for j
  } # for i
# Initial Leave_Action
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl Leave_Action_${i}_${j} (1)"
      } # if
    } # for j
  } # for i

# The Mutex places

  puts "pl Mutex (1)"
  puts "pl Mutex2 (1)"

# All the R_x_x
  for {set i 1} {$i <= $n} {incr i} {
    puts "pl R_${i}_${i} (1)"
  } # for i
```

Continuation…

```
# All the NF_x_y
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl NF_${i}_${j} (1)"
      } # if
    } # for j
  } # for i

# All the NR_x_y
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "pl NR_${i}_${j} (1)"
      } # if
    } # for j
  } # for i

# User "i" makes a "I_Follow_You" on user "j"
# User "j" makes a "You_Follow_Me" on user "i"
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "tr Follow_${i}_${j} R_${i}_${i} NR_${j}_${i}
          Follow_Action_${i}_${j} Mutex ->
          Follow_Action_${i}_${j} Follow_Request_${i}_${j} R_${i}_${i}
          NR_${j}_${i} "
        puts "tr Refuse_${j}_${i} Follow_Request_${i}_${j}
          Refuse_Action_${j}_${i} ->
          Refuse_Action_${j}_${i} Mutex"
        puts "tr Abort_${i}_${j} Follow_Request_${i}_${j}
          Abort_Action_${i}_${j} ->
          Abort_Action_${i}_${j} Mutex"
        puts "tr Accept_${j}_${i} Follow_Request_${i}_${j}
          Accept_Action_${j}_${i} NF_${i}_${j} ->
          Accept_Action_${j}_${i} Accepted_${i}_${j} F_${i}_${j}"
        for {set k 1} {$k <= $n} {incr k} {
          if {!($k == $i)} {
            puts "tr Accepting_${j}_${i}_${k} Accepted_${i}_${j}
              R_${j}_${k} R_${i}_${i} NR_${i}_${k} ->
              NR_${i}_${i} R_${i}_${k} R_${j}_${k}"
            for {set l 1} {$l <= $n} {incr l} {
              if {!($l == $i)} {
                puts "tr Accepting_${j}_${i}_${k} ->
                  Update_Sons_F_${i}_${j}_${k}_${l}"
                puts "tr Update_Sons_F_No_${i}_${j}_${k}_${l}
                  Update_Sons_F_${i}_${j}_${k}_${l} NR_${l}_${i} ->
                  NR_${l}_${i} Update_Sons_${i}_${j}_${k}"
                puts "tr Update_Sons_F_Yes_${i}_${j}_${k}_${l}
                  Update_Sons_F_${i}_${j}_${k}_${l} R_${l}_${i}
                  NR_${l}_${k} ->
                  R_${l}_${k} NR_${l}_${i} Update_Sons_${i}_${j}_${k}"
              } # if
            } # for l
            puts "tr Accept_${j}_${i}_${k}
              Update_Sons_${i}_${j}_${k}*[expr $n - 1] ->
              Mutex"
          } # if
        } # for k
      } # if
    } # for j
  } # for i
```

Continuation…

```
# User "i" makes a "I_Leave_You" on user "j"
# User "j" makes a "I_Leave_You" on user "i"
  for {set i 1} {$i <= $n} {incr i} {
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "tr Leave_${i}_${j} F_${i}_${j} Leave_Action_${i}_${j}
          Mutex ->
          Leave_Action_${i}_${j} Leave_Request_${i}_${j} NF_${i}_${j}"
        for {set k 1} {$k <= $n} {incr k} {
          if {!($k == $i)} {
            puts "tr Leaving_${i}_${j}_${k} Leave_Request_${i}_${j}
              R_${i}_${k} NR_${i}_${i} ->
              NR_${i}_${k} R_${i}_${i} Leave_Update_Root_${i}_${j}_${k}"
              for {set l 1} {$l <= $n} {incr l} {
                if {!($l == $i) && !($l == $k)} {
                  puts "tr Leaving_${i}_${j}_${k} ->
                    Leave_Update_Root_${i}_${j}_${k}_${l}"
                  puts "tr Leave_Update_Root_No_${i}_${j}_${k}_${l}
                    Leave_Update_Root_${i}_${j}_${k}_${l} NR_${l}_${k} ->
                    NR_${l}_${k} Leave_Update_Root_${i}_${j}_${k}"
                  puts "tr Leave_Update_Root_Yes_${i}_${j}_${k}_${l}
                    Leave_Update_Root_${i}_${j}_${k}_${l} R_${l}_${k}
                    NR_${l}_${l} ->
                    NR_${l}_${k} R_${l}_${l}
                    Leave_Update_Root_${i}_${j}_${k}"
              } if
            } # for l
            puts "tr Commit_Leaving_${i}_${j}_${k}
              Leave_Update_Root_${i}_${j}_${k}*[expr $n - 1] ->
              Rebuild_Tree"
          } # if
        } # for k
      } # if
    } # for j
  } # for i
```

Continuation…

```
# SDT rebuilding after a Leave action
  puts "tr Start_Rebuild_Tree Rebuild_Tree ->"
  for {set i 1} {$i <= $n} {incr i} {
    puts "tr Start_Rebuild_Tree -> Rebuild_Tree_${i}"
    puts "tr Rebuild_Tree_No_${i} Rebuild_Tree_${i} NR_${i}_${i} ->
      NR_${i}_${i} End_Rebuild_Tree"
    puts "tr Rebuild_Tree_Maybe_${i} Rebuild_Tree_${i} R_${i}_${i} ->
      R_${i}_${i}"
    for {set j 1} {$j <= $n} {incr j} {
      if {!($i == $j)} {
        puts "tr Rebuild_Tree_Maybe_${i} -> Rebuild_Tree_${i}_${j}"
        puts "tr Rebuild_Tree_No_${i}_${j} Rebuild_Tree_${i}_${j}
          NF_${i}_${j} ->
          NF_${i}_${j} End_Rebuild_Tree_${i}"
        for {set k 1} {$k <= $n} {incr k} {
          if {!($k == $i)} {
            puts "tr Rebuild_Tree_Yes_${i}_${j}_${k}
              Rebuild_Tree_${i}_${j} F_${i}_${j} R_${j}_${k} ->
              F_${i}_${j} R_${j}_${k} Found_Root_${i}_${j}_${k}"
            puts "tr Has_Root_Yes_${i}_${j}_${k} R_${i}_${k}
              Found_Root_${i}_${j}_${k} ->
              R_${i}_${k} End_Rebuild_Tree_${i}"
            puts "tr Has_Root_No_${i}_${j}_${k} Found_Root_${i}_${j}_${k}
              NR_${i}_${k} R_${i}_${i} Mutex2 ->
              R_${i}_${k} NR_${i}_${i}"
            for {set l 1} {$l <= $n} {incr l} {
              if {!($l == $i)} {
                puts "tr Has_Root_No_${i}_${j}_${k} ->
                  Update_Sons_RT_${i}_${j}_${k}_${l}"
                puts "tr Update_Sons_RT_No_${i}_${j}_${k}_${l}
                  Update_Sons_RT_${i}_${j}_${k}_${l} NR_${l}_${i} ->
                  NR_${l}_${i} End_Update_Sons_${i}_${j}_${k}"
                puts "tr Update_Sons_RT_Yes_${i}_${j}_${k}_${l}
                  Update_Sons_RT_${i}_${j}_${k}_${l} R_${l}_${i}
                  NR_${l}_${k} ->
                  NR_${l}_${i} R_${l}_${k}
                  End_Update_Sons_${i}_${j}_${k}"
              } # if
            } # for l
            puts "tr Updated_Sons_${i}_${j}_${k}
              End_Update_Sons_${i}_${j}_${k}*[expr $n - 1] ->
              End_Rebuild_Tree_${i} Mutex2"
          } # if
        } # for k
      } # if
    } # for j
    puts "tr Rebuilt_Tree_${i} End_Rebuild_Tree_${i}*[expr $n - 1] ->
      End_Rebuild_Tree"
  } # for i
  puts "tr Rebuilt_Tree End_Rebuild_Tree*[expr $n] -> Mutex"
# User "j" executes a browsing action
  for {set j 1} {$j <= $n} {incr j} {
    puts "tr Browse_${j} R_${j}_${j} Mutex -> R_${j}_${j}"
    for {set i 1} {$i <= $n} {incr i} {
      if {!($j == $i)} {
        puts "tr Browse_${j} -> Sync_${i}_${j}"
        puts "tr Browse_${i}_${j} R_${i}_${j} Sync_${i}_${j} ->
          R_${i}_${j} End_Sync_${j}"
        puts "tr No_Browse_${i}_${j} NR_${i}_${j} Sync_${i}_${j} ->
          NR_${i}_${j} End_Sync_${j}"
      } # if
    } # for i
    puts "tr End_Browse_${j} End_Sync_${j}*[expr $n - 1] -> Mutex"
  } # for j

} # main

main
```

Figure 59. Script Tcl de génération du réseau de Petri complet

En ce qui concerne le réseau de Petri engendré pour le cas d'une session de navigation coopérative réduite à deux utilisateurs, il est tellement complexe que nous ne le

présenterons pas dans ce document. Il suffit de mentionner qu'il est le produit de la combinaison, en fusionnant les places de même nom, des réseaux de Petri présentés précédemment.

Rappelons finalement dans ce réseau de Petri l'importance des places « Follow_Action_i_j », « Refuse_Action_j_i », « Abort_Action_i_j », « Accept_Action_j_i » et « Leave_Action_i_j ». Selon le marquage de ces places, défini au niveau du marquage initial, les actions « Follow_i_j », « Refuse_j_i », « Abort_i_j », « Accept_j_i » et « Leave_i_j » seront sensibilisées ou non, ce qui nous donnera toute la flexibilité souhaitée au niveau de la vérification formelle de ce modèle global.

III.4. Vérification formelle du modèle de synchronisation de base

Partant des réseaux de Petri génériques introduits dans le paragraphe précédent, nous nous intéressons maintenant à vérifier formellement différentes configurations d'utilisateurs utilisant différentes requêtes de synchronisation en plus de la requête de navigation « Browse ».

Pour ces différentes configurations, nous engendrons, au moyen du script précédemment présenté, le réseau de Petri global, et définissons un état initial par la modification manuelle du réseau engendré. Ensuite nous effectuons sur ce réseau une analyse d'accessibilité au moyen de l'outil TINA. Ensuite, à partir du graphe des marquages obtenu, nous engendrons, au moyen de l'outil ALDEBARAN [Fernandez-96] [CADP], l'automate quotient par rapport à l'équivalence observationnelle qui nous permet de ne visualiser que les actions que nous avons décidé de rendre visibles (les autres actions invisibles correspondent à des actions internes que nous ne souhaitons pas voir apparaître dans l'automate quotient).

III.4.1. Modèle complet avec deux utilisateurs

Nous présentons dans la *Figure 60*, l'automate quotient par rapport à l'équivalence observationnelle pour les hypothèses suivantes :

- nous nous restreignons à deux utilisateurs qui sont asynchrones dans le marquage initial du réseau de Petri ;
- toutes les requêtes de synchronisation et toutes les réponses à ces requêtes de synchronisation sont sensibilisées dans le réseau de Petri engendré ;
- toutes les requêtes de synchronisation, toutes les réponses à ces requêtes de synchronisation, ainsi que l'action « Browse » sont rendues visibles.

L'état 0 représente la situation où les deux utilisateurs sont asynchrones, chacun pouvant donc naviguer de sa propre initiative (i.e. transitions étiquetées par « Browse_1 » et « Browse_2 »). À partir de cet état, un des utilisateurs peut prendre l'initiative de créer une relation de synchronisation (i.e. « Follow_1_2 » ou « Follow_2_1 »), ce qui nous conduit à l'état 3 ou 5, qui représente l'état d'attente de la réponse ou de l'éventuelle annulation ou refus de la requête de synchronisation. Dans le cas d'un refus (i.e. « Refuse_1_2 » ou « Refuse_2_1 » dans les états respectifs) ou d'une annulation (i.e. « Abort_1_2 » ou « Abort_2_1 » dans les états respectifs aussi), l'automate revient à l'état 0. Par contre, si la requête est acceptée (i.e. « Accept_1_2 » ou « Accept_2_1 » respectivamenet), l'automate passe à l'état 1 ou 2, qui représente que la nouvelle relation de synchronisation a bien été créée. Par exemple, dans l'état 1 où l'utilisateur 2 est synchronisé à l'utilisateur 1, nous pouvons voir que l'utilisateur 2 ne peut pas naviguer de sa propre initiative (i.e. il n'existe pas de transition étiquetée par l'action « Browse_2 »), et

que sa navigation est maintenant synchronisée à celle de l'utilisateur 1 (i.e. transitions étiquetées par « Browse_1 » puis par « Browse_2_1 »). Finalement, si l'automate se trouve dans l'état 1 ou 2, et si l'un des utilisateurs désire supprimer la relation de synchronisation (i.e. transitions étiquetées par les actions « Leave_2_1 » ou « Leave_1_2 »), l'automate revient à l'état initial 0.

Il est facile de vérifier que le comportement décrit par l'automate quotient de la *Figure 60* correspond effectivement au service de synchronisation que nous souhaitons offrir avec notre système de navigation coopérative, dans le cas de deux utilisateurs.

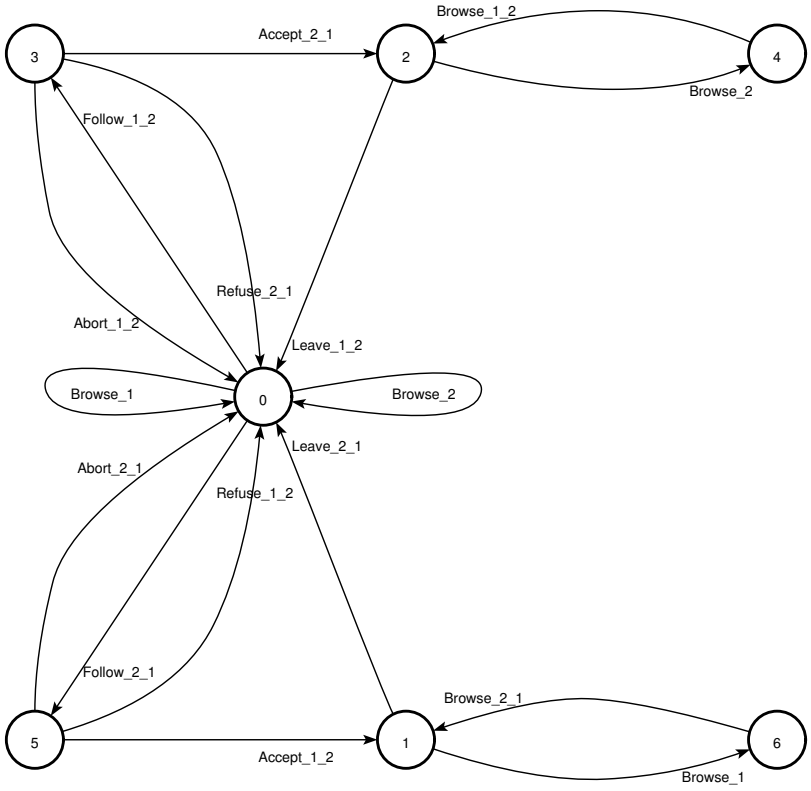


Figure 60. Automate quotient complet pour deux utilisateurs

III.4.2. Modèle complet avec trois utilisateurs

Nous présentons dans la *Figure 61*, l'automate quotient par rapport à l'équivalence observationnelle avec les mêmes hypothèses que celles définies pour la *Figure 60*, sauf que maintenant nous considérons une session de navigation coopérative comprenant initialement trois utilisateurs asynchrones au lieu de deux.

Cette représentation de l'automate quotient est déjà peu lisible pour 3 utilisateurs à cause de l'explosion combinatoire due à l'entrelacement des différentes requêtes et réponses de synchronisation, ainsi que de l'action « Browse ».

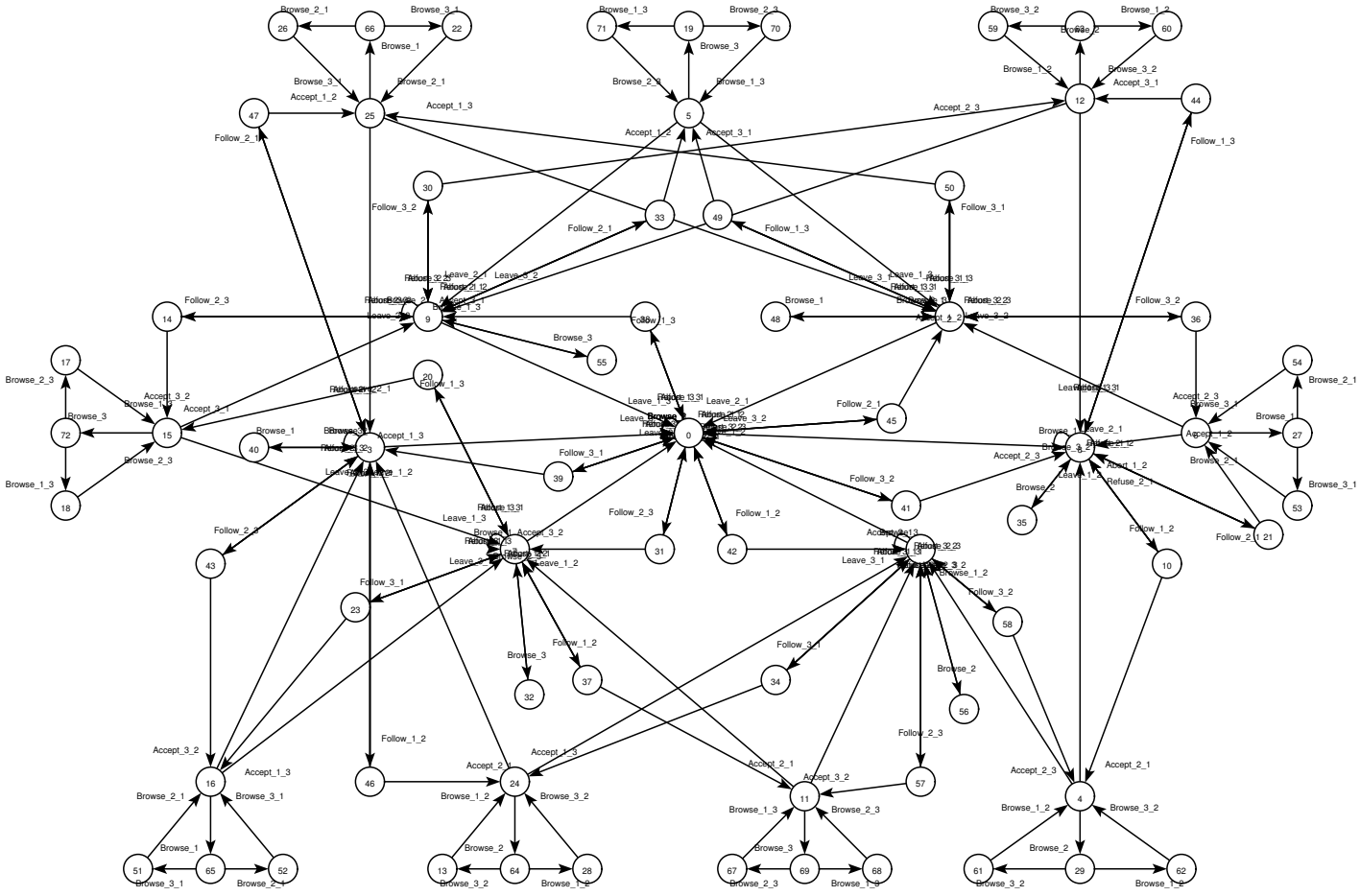


Figure 61. Automate quotient complet pour trois utilisateurs

III.4.3. Modèles partiels avec trois, quatre et cinq utilisateurs

Dans le but de présenter des automates quotients plus lisibles, nous considérons différents scénarios caractéristiques du fonctionnement du système de navigation coopérative avec trois, quatre et cinq utilisateurs.

III.4.3.1. Scénario 1 : trois utilisateurs asynchrones, « Follow_2_1 », « Leave_2_1 » et « Browse »

Pour la modélisation de ce Scénario 1, nous procédons aux simplifications suivantes :

- au lieu de considérer toutes les requêtes de synchronisation « Follow_i_j », nous ne considérons que la requête « Follow _2_1 », ce qui est réalisé en ne marquant dans le marquage initial du réseau de Petri que la place « Follow_Action_2_1 » (et non pas toutes les places « Follow_Action_i_j ») ; cette action « Follow _2_1 » est également toujours considérée comme étant visible.
- d'une manière similaire, nous ne considérons que les requêtes de synchronisation « Abort_2_1 » et « Leave_2_1 », ainsi que les réponses aux requêtes de synchronisation « Refuse_1_2 » et « Accept_1_2 », ceci étant toujours réalisé par le marquage initial des places correspondantes du réseau de Petri ; ces requêtes de synchronisation et ces réponses aux requêtes de synchronisation sont également toujours considérées comme étant visibles.

L'automate quotient correspondant à ces hypothèses et pour lequel l'action de navigation « Browse » est considérée comme étant toujours visible est présenté dans la *Figure 62*.

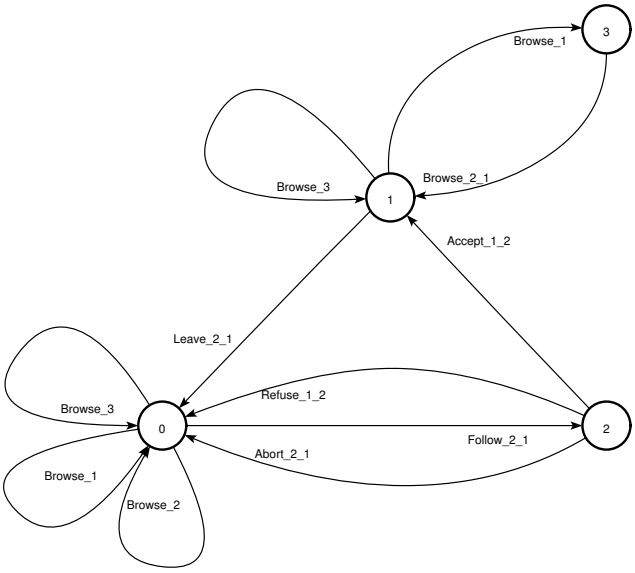


Figure 62. Automate quotient pour le scénario 1

Quand l'automate se trouve dans l'état 0, chaque utilisateur asynchrone peut naviguer de sa propre initiative (« Browse_1 », « Browse_2 » et « Browse_3 »). L'occurrence d'une requête de synchronisation « Follow_2_1 » suivie de l'occurrence d'une acceptation de cette requête « Accept_1-2 » fait passer l'automate dans l'état 1, où la navigation de l'utilisateur 2 est maintenant synchronisé à celle de l'utilisateur 1 (action « Browse_1 » suivie de « Browse_2_1 », l'utilisateur 3 étant toujours asynchrone (action « Browse_3 » (notons que nous n'avons pas d'entrelacement complet entre les actions de navigation « Browse_3 » d'une part et « Browse_1 » et « Browse_2_1 » d'autre part, car nous avons introduit dans notre modélisation du réseau de Petri (*Figure 56*), une place Mutex pour limiter le parallélisme). Bien évidemment, lorsque la relation de synchronisation est supprimée (occurrence de l'action « Leave_2_1 »), l'automate revient à l'état 0.

III.4.3.2. Scénario 2 : trois utilisateurs appartenant à un même SDT, « Leave_2_1 » et « Browse »

Considérons maintenant le cas avec trois utilisateurs, sachant que l'utilisateur 3 est synchronisé à l'utilisateur 2 qui est lui même synchronisé à l'utilisateur 1 qui est asynchrone et analysons l'effet d'une requête de synchronisation « Leave_2_1 » dans l'automate quotient de la *Figure 63*.

Dans cet exemple nous pouvons observer que dans l'état initial (état 0), le seul utilisateur à pouvoir naviguer de sa propre initiative est l'utilisateur 1 (« Browse_1 »), la navigation des deux autres utilisateurs étant synchronisée à celle de l'utilisateur 1 (actions « Browse_2_1 » et « Browse_3_1 » qui sont entrelacées car se produisant en parallèle une fois que l'action Browse_1 a été réalisée). Toujours à partir de l'état 0, si un des utilisateurs concernés exécute l'action « Leave_2_1 », alors l'automate passe à l'état 1, où les utilisateurs 1 et 2 peuvent naviguer de leur propre initiative, sachant que la navigation de l'utilisateur 3 est maintenant synchronisée à celle de l'utilisateur 2.

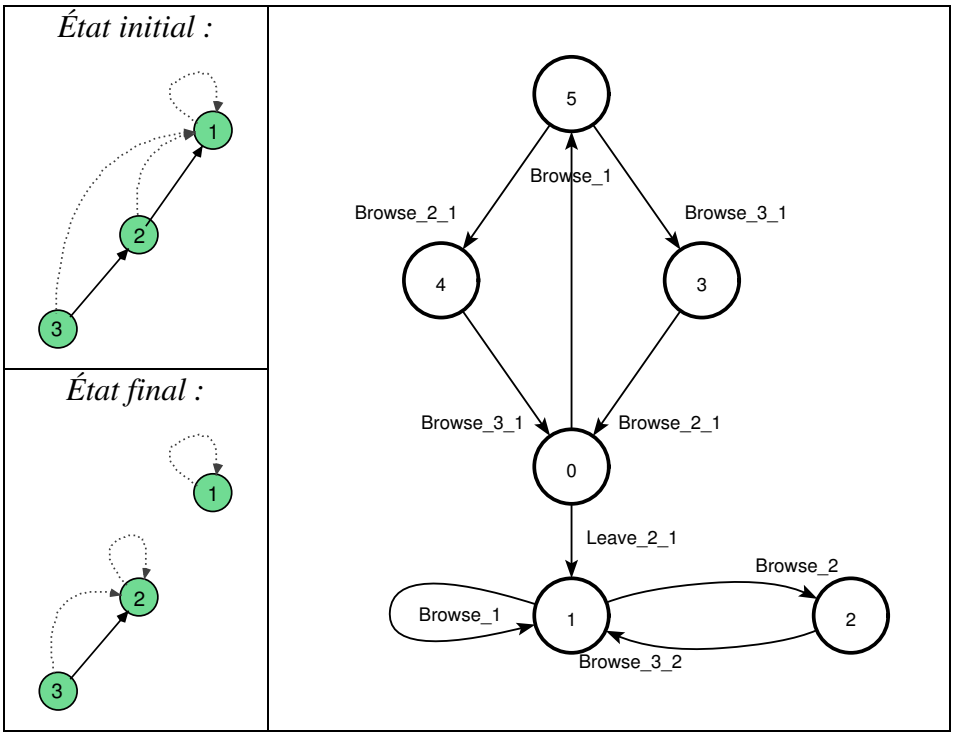


Figure 63. Automate quotient pour le scénario 2

III.4.3.3. Scénario 3 : quatre utilisateurs appartenant à un même SDT, « Leave_2_1 » et « Browse »

Nous considérons maintenant le cas d'une session de navigation coopérative composée de 4 utilisateurs et présentons un ensemble de relations de synchronisation comme nous pouvons le voir ci-dessous. Comme précédemment, nous analysons dans cette configuration à 4 utilisateurs, l'effet d'une requête de synchronisation « Leave_2_1 » dans l'automate quotient de la *Figure 64*.

Dans cet exemple, de manière analogue aux exemples précédents, lorsqu'on part de l'état 0 le seul utilisateur à pouvoir naviguer de sa propre initiative est l'utilisateur 1 (« Browse_1 »), tous les autres utilisateurs étant, par définition, synchronisés à cet utilisateur 1. L'occurrence de la requête de synchronisation « Leave_2_1 » conduit l'automate à l'état 1, où les deux utilisateurs pouvant naviguer de leur propre initiative sont maintenant les utilisateurs 1 et 2.

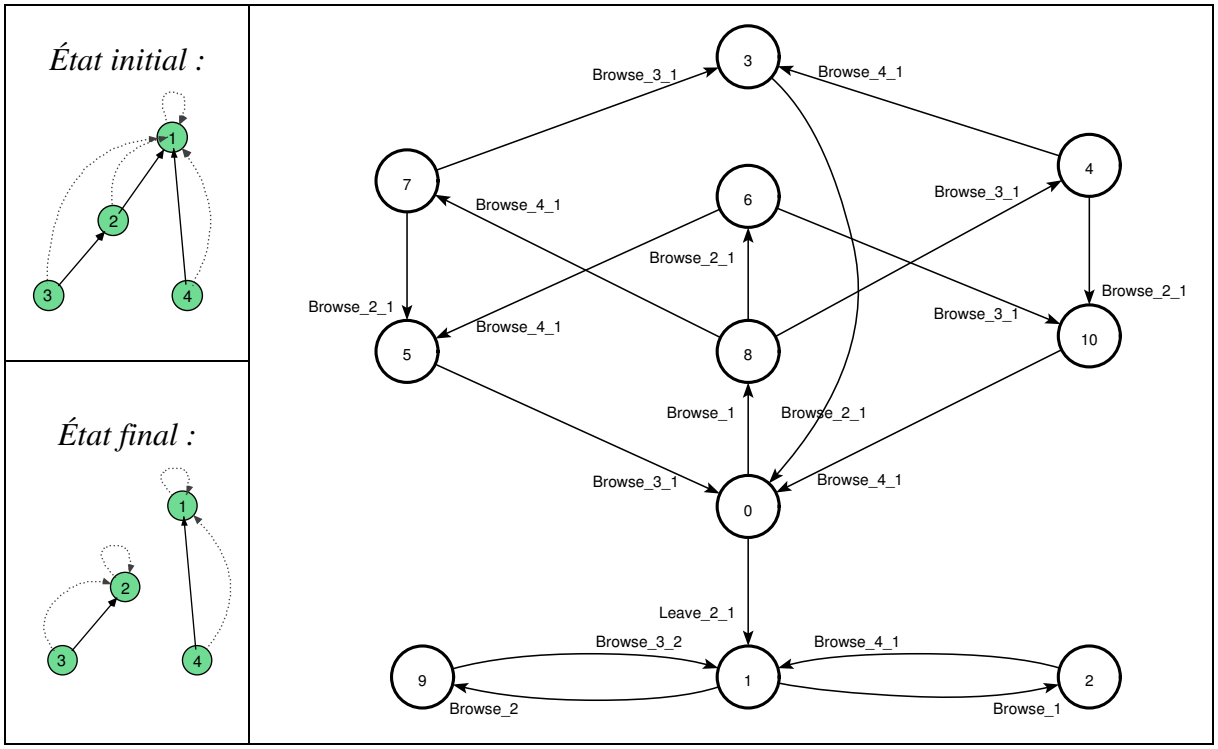


Figure 64. Automate quotient pour le scénario 3

III.4.3.4. Scénario 4: cinq utilisateurs appartenant à deux SDTs, « Follow_2_1 » et « Browse »

L'automate quotient associé au scénario 4 est illustré dans la *Figure 65*. Ce scénario correspond au cas d'une session constituée de cinq utilisateurs, avec deux SDTs, l'un synchronisant deux utilisateurs, les autres trois utilisateurs. Nous analysons pour cette configuration l'effet d'une requête de synchronisation « Follow_2_1 ».

A titre de conclusion, nous pouvons indiquer que nous avons analysé plusieurs configurations (configurations exhaustives pour deux et trois utilisateurs et configurations non exhaustives au delà de trois utilisateurs) et que pour chacune de ces configurations nous avons analysé l'automate quotient résultant. Ces analyses nous ont convaincu que pour ces différentes configurations, les mécanismes de synchronisation modélisés offrent bien le service de synchronisation attendu.

Au delà de trois utilisateurs, nous n'avons pas été en mesure d'analyser des configurations exhaustives, à cause de l'explosion combinatoire induite par notre composant « Rebuild_Tree » suite à la suppression d'une relation de synchronisation.

Pour donner une idée approximative de la complexité des réseaux de Petri engendrés, dans le tableau de la *Figure 66* nous présentons une synthèse de résultats pour différents nombres d'utilisateurs, notamment le nombre de places et de transitions des réseaux de Petri et le nombre de marquages du réseau et le nombre d'états de l'automate de quotient minimisé.

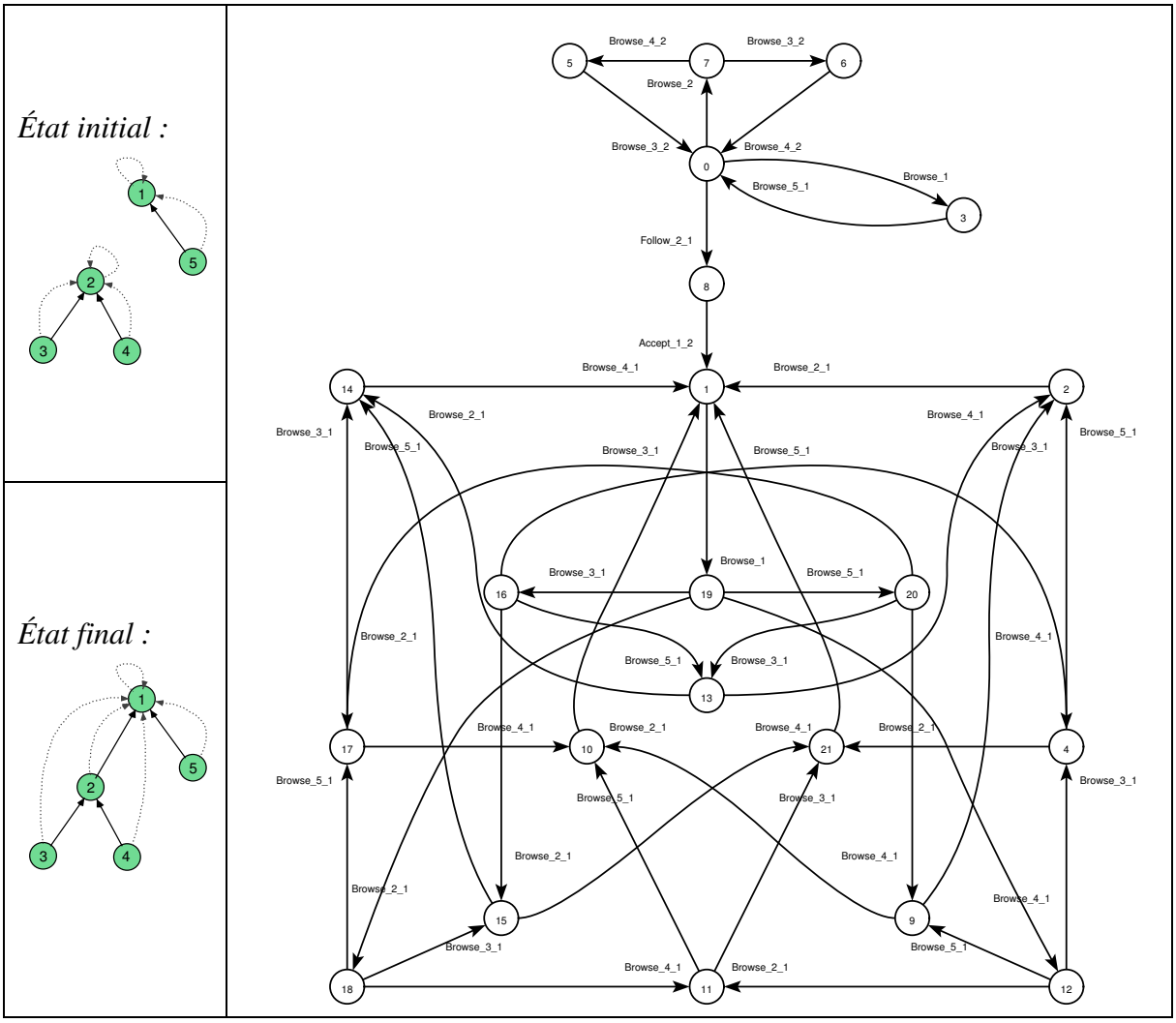


Figure 65. Automate quotient pour le scénario 4

Nombre d'utilisateurs	Nombre de places et transitions du réseau de Petri	Cas spécifique	Taille du graphe de marquages	Taille de l'automate quotient après minimisation
Deux	54 places 52 transitions	***Figure 60*** (complet)	40 marquages 58 transitions	7 états 16 transitions
Trois	211 places 281 transitions	***Figure 61*** (complet)	4007 marquages 13022 transitions	73 états 186 transitions
		Figure 62 (Follow_2_1 & Leave_2_1)	248 marquages 694 transitions	4 états 11 transitions
		Figure 63 (Leave_2_1)	594 marquages 2004 transitions	6 états 9 transitions
Quatre	624 places 982 transitions	***Figure 64*** (Leave_2_1)	109232 marquages 717811 transitions	11 états 18 transitions
Cinq	1509 places 2587 transitions	***Figure 65*** (Follow_2_1)	68 marquages 138 transitions	22 états 42 transitions

Figure 66. Tableau comparatif des analyses faites sur les réseaux de Petri

III.5. Le modèle de synchronisation étendu

Dans les paragraphes précédents nous avons présenté le modèle de synchronisation de base pour notre proposition de navigation coopérative et avons montré mis en évidence plusieurs résultats de vérification formelle validant les mécanismes de synchronisation proposés. Ce modèle de base s'appuie, au niveau du mécanisme de création de relations de synchronisation, sur un mécanisme d'autorisation préalable.

Nous proposons dans ce paragraphe d'étendre le modèle de synchronisation de base, en introduisant une notion de rôle et une notion de privilège associé à un rôle, dans le but de généraliser les différentes actions de synchronisation présentées jusqu'à maintenant.

Deux extensions au mécanisme de création de relations de synchronisation nous ont paru intéressantes :

- permettre à un utilisateur jouant un rôle particulier d'espionner les actions de navigation d'un autre utilisateur jouant un autre rôle ;
- permettre à un utilisateur jouant un rôle particulier de forcer un autre utilisateur (ou un ensemble d'utilisateurs) jouant un autre rôle, à se synchroniser avec lui.

La notion de rôle est fondamentale, car conceptuellement chaque rôle représente une catégorie d'utilisateurs, c'est à dire un ensemble d'utilisateurs partageant les mêmes droits de synchronisation. Caractériser des utilisateurs par l'utilisation de rôles facilite ainsi les tâches liées à la gestion de la session de navigation coopérative [Wang-99].

Les relations de synchronisation entre utilisateurs sont normalement créées sur la base d'un protocole d'autorisation : un utilisateur qui veut créer une relation de synchronisation avec un autre utilisateur doit tout d'abord lui demander son accord. Si la demande est acceptée la relation de synchronisation est créée, autrement non.

Nous pouvons imaginer qu'il existe des privilèges entre utilisateurs selon le rôle qu'ils jouent dans une session de navigation coopérative. Ces privilèges permettent d'échapper au mécanisme d'autorisation pour créer des relations de synchronisation, de telle façon que, dans certaines conditions, il est possible de créer des relations de synchronisation de manière inconditionnelle.

Les deux types de privilège envisagés correspondent aux deux prédicats suivants :

- *canSpy* : représente la possibilité pour un utilisateur, disposant d'un rôle avec ce privilège, de synchroniser de manière inconditionnelle ses activités de navigation à celles d'un autre utilisateur, sans que ce dernier ne se puisse s'en rendre compte : l'utilisateur sur lequel l'on utilise ce privilège est en fait espionné ; (voir ci-dessous l'action de synchronisation « I_Spy_You » ;
- *canForce* : représente la possibilité pour un utilisateur, disposant d'un rôle avec ce privilège, de faire en sorte qu'un ou plusieurs utilisateurs disposant d'un rôle sur lequel s'applique ce privilège, se synchronise(nt) avec lui de manière inconditionnelle ; (voir ci-dessous l'action « You_Join_Me »).

Évidemment si aucun privilège n'est défini lors de la configuration de la session, toutes les opérations de synchronisation entre utilisateurs restent régies par le mécanisme d'autorisation mentionné auparavant. Par ailleurs, deux utilisateurs de rôle différent ayant entre eux des privilèges peuvent bien entendu continuer à utiliser les mécanismes de synchronisation du modèle de base qui dépendent d'une procédure d'autorisation.

Ceci nous amène à proposer la définition suivante pour une session de navigation coopérative:

$$S = (SID, U, R, role, canSpy, canForce)$$

où

- *SID* est un identificateur de session
- $U = \{u_i\}, i \in [1, N]$ est l'ensemble des utilisateurs connectés à la session ;
- $R = \{r_j\}, j \in [1, M]$ est l'ensemble de rôles définis pour la session ;
- $P = \{p_k\}, k \in [1, L]$ est l'ensemble des privilèges définis entre certains rôles ;
- $role : U \rightarrow R$ est une fonction définissant le rôle d'un utilisateur ;
- $canSpy : R \times R \rightarrow \{T, F\}$ est un prédicat qui donne vrai quand un utilisateur de rôle *R1* peut espionner un utilisateur de rôle *R2* ;
- $canForce : R \times R \rightarrow \{T, F\}$ est un prédicat qui donne vrai quand un utilisateur de rôle *R1* peut forcer la synchronisation d'un utilisateur de rôle *R2*.

III.5.1. L'action de synchronisation « I_Spy_You »

L'action de synchronisation « I_Spy_You » est une extension de l'action de synchronisation « I_Follow_You ». Elle permet à un utilisateur *i* de se synchroniser à un utilisateur *j* sans autorisation préalable de *j*, et sans que l'utilisateur *j* ne puisse s'en rendre compte. Soient *i* et *j*, deux utilisateurs de rôle respectif *R1* et *R2*. Si *canSpy(R1, R2)* est vrai, alors *i* peut espionner *j* au moyen d'une action « I_Spy_You », c'est-à-dire que *i* peut synchroniser sa navigation à celle de *j*, sans que *j* ne soit conscient de cette synchronisation. Par voie de conséquence, l'utilisateur *j* ne dispose également plus de la possibilité, au moyen de un « I_Leave_You », de supprimer la relation de synchronisation que *i* a établie avec lui.

III.5.2. L'action de synchronisation « You_Join_Me »

L'action de synchronisation « You_Join_Me » est une extension à l'action de synchronisation « You_Follow_Me ». Elle permet à un utilisateur *i* de forcer de qu'un utilisateur *j* (ou un ensemble d'utilisateurs) se synchronise avec lui de manière inconditionnelle et donc sans autorisation préalable. Soient *i* et *j*, deux utilisateurs de rôle respectif *R1* et *R2*. Si *canForce(R1, R2)* est vrai, alors *i* peut forcer un utilisateur *j* de rôle *R2* (ou un ensemble d'utilisateurs *J* de rôle *R2*) de se synchroniser avec lui au moyen d'une action « You_Join_Me ». L'utilisateur *j* est ainsi synchronisé à *i* et a conscience de cette synchronisation. Cette synchronisation est inconditionnelle et supprime par conséquent toute relation de synchronisation établie préalablement pour l'utilisateur *j* (ceci est une conséquence de nos hypothèses de cohérence, à savoir qu'un utilisateur ne peut, à un instant donné, être synchronisé qu'à un seul autre utilisateur). D'une manière similaire, cette (ou ces) relation(s) de synchronisation ne pourront être supprimées qu'à l'initiative de l'utilisateur *i* (au moyen d'une action « I_Leave_You(j) » pour un utilisateur, ou « I_Leave_You(J) » pour un ensemble d'utilisateurs.

III.5.3. Formalisation du modèle de synchronisation étendu

Nous proposons dans ce paragraphe une formalisation du modèle de synchronisation étendu en nous restreignant à une modélisation au moyen d'automates étendus. Notre but ici est de spécifier formellement les nouvelles actions de synchronisation que nous avons définies, mais pas de les vérifier formellement. Nous ne présentons ci-dessous que les ajouts au modèle de synchronisation de base défini dans le paragraphe III.3.1.

Requêtes de synchronisation :

- *I_Spy_You* : un utilisateur espionne un autre utilisateur en se synchronisant avec lui sans autorisation préalable ;

- *You_Join_Me* : un utilisateur force un (ou plusieurs) utilisateur(s) à se synchroniser avec lui.

Prédicats sur l'ensemble des rôles :

- *canSpy(R1, R2) : R × R → {T, F}* est un prédicat qui donne vrai si tout utilisateur de rôle *R1* a le privilège d'espionner tout utilisateur de rôle *R2* ;
- *canForce(R1, R2) : R × R → {T, F}* est un prédicat qui donne vrai si tout utilisateur de rôle *R1* a le privilège de forcer tout utilisateur de rôle *R2* à se synchroniser avec lui.

Fonction assurant la correspondance entre utilisateurs et rôles :

- *role(x)* : cette fonction retourne le rôle d'un utilisateur *x*.

Fonctions sur la structure de données caractérisant l'ensemble des SDTs de la session:

- *updateSDT-add(mode, i, j)* : cette fonction actualise l'ensemble des SDTs associés à la session de navigation coopérative, suite à l'ajout d'une relation de synchronisation entre les utilisateurs *i* et *j* (la navigation de *i* suit la navigation de *j*) en utilisant une relation de synchronisation de type *mode* ;
- *updateSDT-remove(mode, i, j)* : cette fonction actualise l'ensemble des SDTs associés à la session de navigation coopérative, suite à la suppression d'une relation de synchronisation entre les utilisateurs *i* et *j* (*i* devient ainsi asynchrone et peut naviguer de sa propre initiative) en utilisant une relation de synchronisation de type *mode*.

Partant de ces notations, nous obtenons l'automate étendu que nous présentons dans la *Figure 67*, dans lequel nous présentons les possibles état de synchronisation dont l'utilisateur *i* peut être soumis.

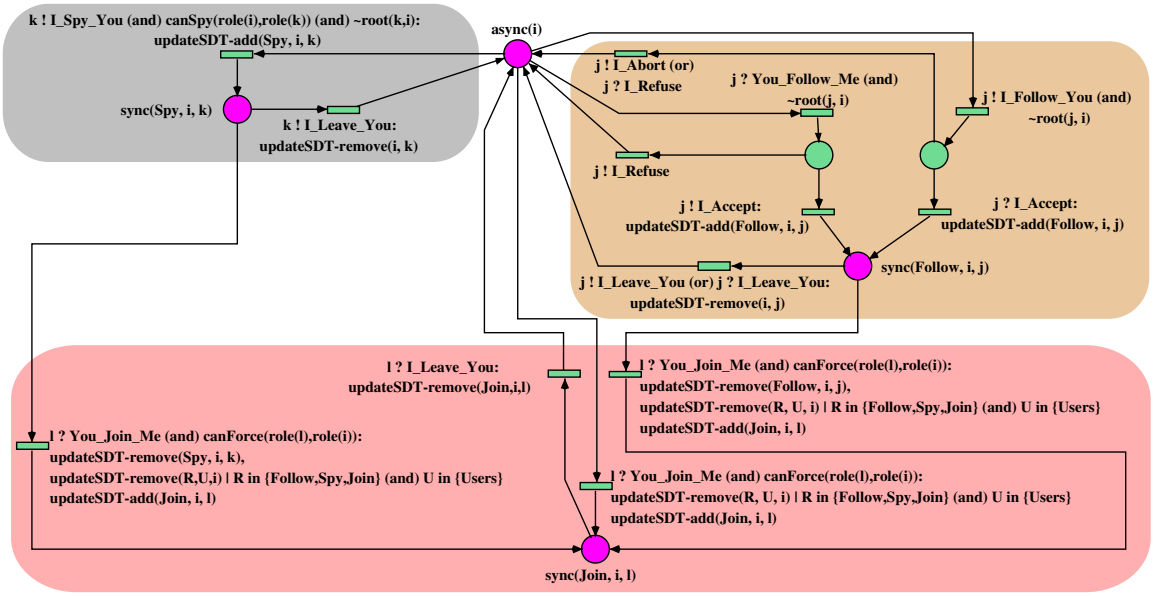


Figure 67. Synchronisation d'un utilisateur i avec l'utilisateur j/k/l (modèle de synchronisation étendu)

Dans ce nouveau modèle, nous proposons que l'état de synchronisation entre deux utilisateurs fasse référence au type de relation (Follow, Spy, Join), de telle façon qu'il soit toujours possible de connaître ce type de relation.

L'aire colorée en haut à droite correspond à la partie de l'automate étendant l'automate présenté dans la *Figure 41*. La modélisation du comportement des requêtes de

synchronisation « I_Spy_You » et « You_Join_Me » est représentée dans les aires colorées en haut à gauche, et en bas respectivement.

Le cas de la requête de synchronisation « You_Join_Me » est un plus complexe, car tous les états de synchronisation sont concernés. Ceci est dû au fait qu'une requête « You_Join_Me » peut être envoyée à tout un autre utilisateur, indépendamment de son état de synchronisation courant. Le seul cas où cette requête de synchronisation ne peut être appliquée correspond au cas où l'utilisateur, dont on prétend forcer la synchronisation, est déjà synchronisé (au moyen d'une requête « You_Join_Me ») à un autre utilisateur jouant un rôle lui permettant de disposer de ce privilège.

Lors de l'exécution d'une requête de synchronisation « You_Join_Me », toutes les relations de synchronisation dont chaque utilisateur appelé à se synchroniser appartenait jusqu'à ce moment-là sont systématiquement éliminées avant de créer la nouvelle relation de synchronisation.

Dans la *Figure 68*, nous pouvons observer le comportement de l'utilisateur *i* avec lequel un autre utilisateur *k* désire se synchroniser. Cet automate ressemble à l'automate précédent et doit pouvoir être analysé par le lecteur sans trop de difficultés. En fusionnant par places communes les automates étendus présentés dans la *Figure 67* et *la Figure 68*, nous obtenons la description de l'ensemble des comportements possibles du point de vue de l'utilisateur *i*.

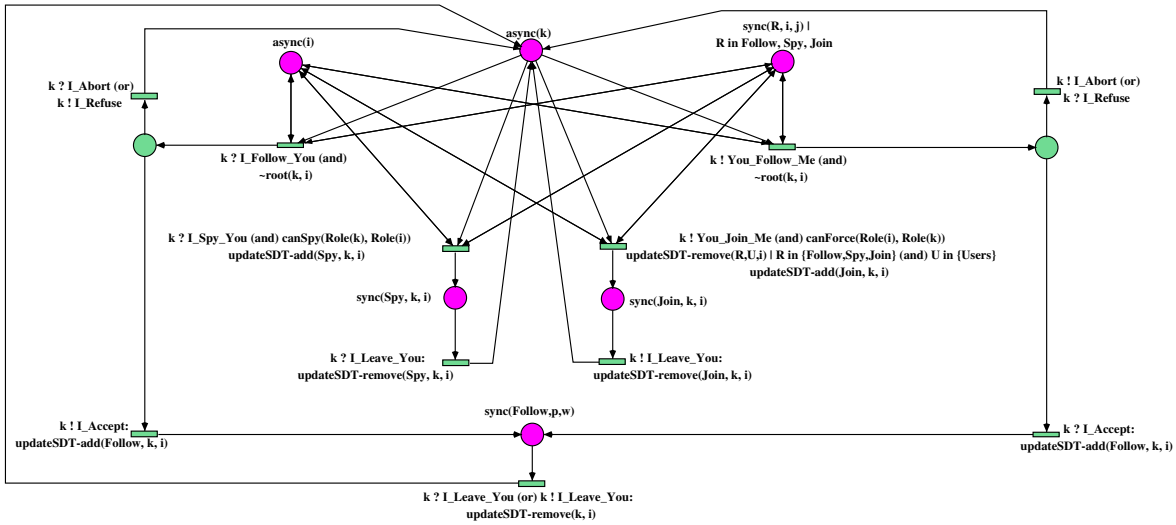


Figure 68. Synchronisation d'un utilisateur k avec l'utilisateur i (modèle de synchronisation étendu)

III.6. Conclusion

Dans ce chapitre nous avons présenté la formalisation du modèle de synchronisation pour notre système de navigation coopérative, et avons démontré qu'il est cohérent (i.e. il n'admet pas la création d'états de synchronisation incohérents).

Nous avons fait une modélisation complète du modèle de synchronisation de base à deux niveaux : modélisation du comportement local d'un utilisateur au moyens d'automates étendus, modélisation globale, au moyen d'un réseau de Petri, d'un ensemble d'utilisateurs pouvant se synchroniser au sein d'une session. Cette modélisation inclut la modélisation du SDT et les différentes opérations (création et suppression des relations de synchronisation) réalisées sur le SDT au moyen de places et de transitions.

Nous avons montré comment engendrer de manière systématique les réseaux de Petri correspondant aux différentes situations que nous souhaitions analyser, au moyen de scripts appropriés. La vérification formelle a consisté à engendrer de manière automatique les automates quotients (vis à vis de l'équivalence observationnelle) caractérisant les services correspondants à ces différentes situations, et prouvant ainsi de manière intuitive que le service rendu correspond effectivement au service attendu.

Nous avons ensuite montré comment généraliser le modèle de synchronisation de base en introduisant de nouvelles requêtes de synchronisation s'appuyant sur la notion de rôle et avons proposé leur formalisation (modèle local à base d'automates étendus).

S'appuyant sur cette formalisation nous allons aborder dans les chapitres suivants l'architecture puis l'implémentation et le déploiement du système de navigation coopérative proposé.

Chapitre IV

Architecture du système de navigation coopérative

IV.1. Introduction

Un système de navigation coopérative a pour objectif principal de permettre à des utilisateurs enregistrés dans une session coopérative de synchroniser leurs activités de navigation sur le Web. L'intégration des mécanismes de synchronisation et de navigation dépend de la manière dont l'architecture globale du système est définie. Idéalement, ce système de navigation coopérative doit également pouvoir s'intégrer à d'autres outils de communication et de coopération, par exemple un système de messagerie instantanée (« chat ») ou un système de vidéoconférence pour permettre aux utilisateurs d'une même session de navigation coopérative de communiquer entre eux de manière informelle.

Dans ce chapitre, nous introduisons et développons l'architecture du système de navigation coopérative que nous proposons. Dans un premier paragraphe, nous présentons une vue globale du système en décrivant de manière informelle les principaux modules de l'architecture et la manière dont ils interagissent. Dans le paragraphe suivant, nous introduisons une méthode de modélisation d'architecture s'appuyant sur le profil UML/SDL [UML] [SDL] [Björkander-03] [Björkander-00a] [Björkander-00b] supporté par l'outil Tau G2 de Telelogic [Telelogic]. Dans un troisième paragraphe nous appliquons cette méthode à la formalisation de l'architecture du système de navigation coopérative. Dans un quatrième paragraphe, nous montrons finalement comment valider l'architecture proposée.

IV.2. Vue générale du système de navigation coopérative

Dans un système de navigation coopérative, les deux fonctionnalités les plus importantes sont la synchronisation des utilisateurs et la synchronisation de la navigation. La première fonctionnalité comprend l'ensemble des requêtes des utilisateurs destinées à créer et supprimer des relations de synchronisation entre eux, et la seconde concerne le traitement des requêtes HTTP pour le rapatriement de ressources à partir de serveurs Web distants. Ces deux fonctionnalités sont bien évidemment fortement couplées entre elles, les requêtes HTTP devant se synchroniser selon les relations de synchronisation établies entre les utilisateurs (le modèle formel de synchronisation des utilisateurs a été présenté dans le chapitre précédent).

Pour satisfaire ces objectifs, le système de navigation coopérative est un système complexe composé de différents modules. La *Figure 69* présente une vue générale de

l'architecture du système de navigation coopérative en présentant ses principaux modules et la manière dont ils interagissent entre eux et avec l'extérieur.

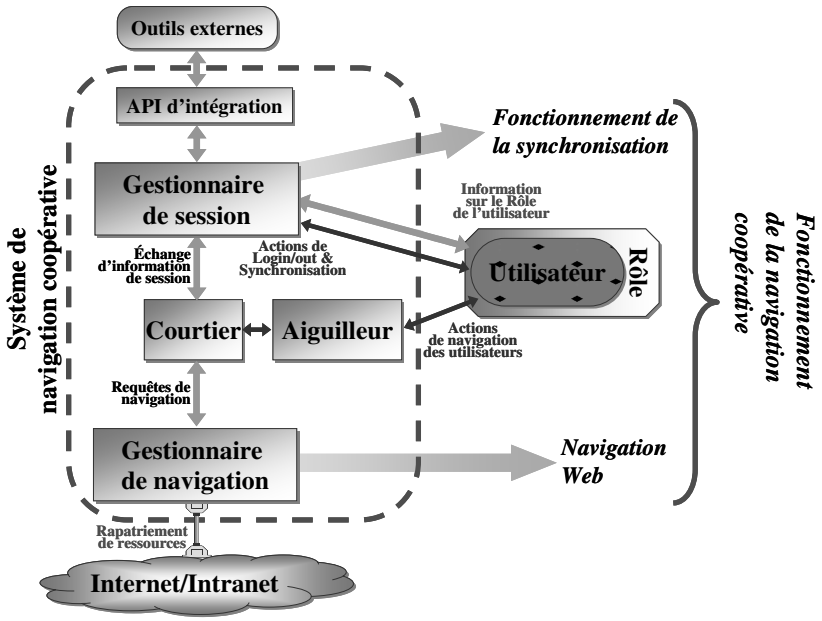


Figure 69. Vue générale du système de navigation coopérative

Dans ce diagramme apparaissent les deux modules principaux de l'architecture, à savoir : le gestionnaire de session et le gestionnaire de navigation, qui sont interconnectés par un troisième composant, le courtier, qui est lui-même responsable de la réception des requêtes de navigation en provenance des utilisateurs et de la transmission des réponses à ces requêtes. Le courtier permet ainsi de coordonner les activités des gestionnaires de session et de navigation.

Le gestionnaire de session a pour but de gérer globalement la session de navigation coopérative, y compris tout ce qui concerne la synchronisation des utilisateurs. Parmi ses responsabilités, nous pouvons identifier :

- la gestion et la mise à jour de l'état de synchronisation des utilisateurs appartenant à une même session de navigation coopérative selon le modèle décrit dans le chapitre précédent ;
- la prise de décision concernant l'accessibilité d'une ressource en fonction de l'état de synchronisation de l'utilisateur qui en a fait la requête ;
- la synchronisation de la navigation lorsque c'est nécessaire ;
- l'intégration éventuelle du système de navigation coopérative à d'autres outils de communication et de coopération externes.

Le gestionnaire de navigation a pour but de prendre en charge le traitement des requêtes HTTP engendrées par les utilisateurs au cours d'une session de navigation coopérative. Il est ainsi capable de recevoir ces requêtes et de rapatrier les ressources associées.

Ces différents modules interagissent d'une manière bien définie, comme cela est présenté dans la *Figure 70*. Remarquons que ici, dans le but de simplifier la description des interactions, nous ne faisons pas référence à l'aiguilleur, qui sera introduit ultérieurement.

Dans un premier temps, l'utilisateur asynchrone exprime une requête de rapatriement d'une ressource (1), qui est directement traitée par le courtier. Celui-ci demande au gestionnaire de session si cette requête peut être satisfaite ou non (2). Si la réponse du gestionnaire de session est affirmative, le courtier envoie la requête au gestionnaire de navigation (3). Dans ce gestionnaire, la requête est reçue par le module de

rapatriement, qui demande au module de stockage local si la ressource a déjà été rapatriée, et est donc dans le cache local (4). Si ce n'est pas le cas, la ressource est directement rapatriée d'un serveur Web (5-6), et si cette ressource est identifiée comme étant une ressource au format HTML, elle est envoyée au module de traduction (7). Une fois que la ressource a été traduite, elle est retournée au module de rapatriement (8), ainsi qu'au module de cache local (9). À ce moment là le module de rapatriement envoie la réponse au courtier (10), et la réponse est envoyée à l'utilisateur (11).

Une fois que cette procédure est terminée, le courtier demande au gestionnaire de session de synchroniser les navigateurs des utilisateurs qui ont établi des relations de synchronisation avec l'utilisateur en question (12). Le gestionnaire de session envoie un message au navigateur de tous ces utilisateurs (13). En réponse, chaque navigateur envoie de manière indépendante une requête HTTP pour rapatrier la ressource indiquée (14). Chaque requête arrive directement au courtier qui interagit comme précédemment avec le gestionnaire de navigation (15) qui lui-même interagit avec le module de cache local (16). La ressource est alors récupérée du cache local (17), retournée au module de rapatriement (18) qui l'envoie au courtier (19) qui l'envoie finalement à l'utilisateur (20).

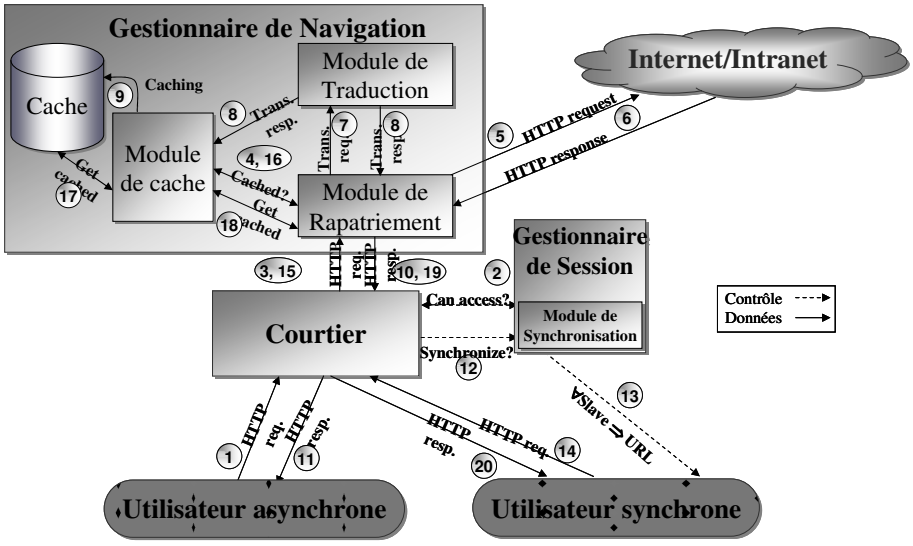


Figure 70. Synchronisation de la navigation

IV.3. Modélisation dans le profil UML/SDL

Dans le but de modéliser l'architecture et le fonctionnement du système de navigation coopérative, nous avons choisi le langage UML (Unified Modeling Language – langage de modélisation unifié) et plus spécifiquement le profil UML/SDL supporté par l'outil Tau G2 de Telelogic. Nous allons introduire ce profil avant de détailler la méthode employée pour modéliser et valider l'architecture du système de navigation coopérative proposé.

IV.3.1. Introduction au profil UML/SDL

UML est un langage graphique destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser 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 (Object Mangement Group) [OMG], un

consortium de plus de 800 sociétés et universités actives dans le domaine des technologies objets.

UML 2.0 propose 13 diagrammes qui permettent de définir autant de points de vue d'un même système (à logiciel prépondérant). Ces diagrammes peuvent être classés en deux catégories qui correspondent respectivement à des vues statiques (*Figure 71*), et à une représentation de la dynamique des systèmes (*Figure 72.*).

Type de diagramme statique	Utilisation
Diagramme de cas d'utilisation	Ce diagramme sert à délimiter le périmètre du système modélisé et à identifier les relations entre les fonctionnalités ou services offerts par ce système et des acteurs externes.
Diagramme de classes	Ce diagramme représente la structure statique du système en identifiant les types des objets sous forme de classes et en précisant les relations (composition, agrégation et héritage) entre ces classes. Ce diagramme peut être utilisé à des fins d'analyse et surtout pour la phase de la conception.
Diagramme d'objets	Ce diagramme permet d'illustrer des configurations particulières d'objets en complément au diagramme de classes.
Diagramme de structure composite	Ce diagramme est utilisé dans un processus de décomposition d'une classe qui forme un tout pour exprimer les communications entre les parties (parts en anglais) qui forment ce tout.
Diagramme de paquetages	Ce diagramme permet de regrouper des classes dans des unités logiques appelées paquetages.
Diagramme de composants	Ce diagramme permet de regrouper des objets en composants logiciels.
Diagramme de déploiement	Ce diagramme décrit la répartition sur des sites physiques géographiquement distribués des composants logiciels identifiés dans le diagramme de composants.

Figure 71. Diagrammes statiques de UML

Type de diagramme dynamique	Utilisation
Diagramme d'activité	Ce diagramme décrit le flux de donnée et/ou de contrôle d'un comportement (similaire à un organigramme).
Diagramme de machines à états	Ce diagramme permet de décrire le comportement interne d'un objet et sa réactivité à l'environnement sous la forme d'une machine à états.
Diagramme de vue globale	Ce diagramme sert à structurer les scénarios définis par les diagrammes de séquences.
Diagramme de séquences	Ce diagramme décrit sous forme de scénarios un ordonnancement possible d'interactions entre objets qui composent le système.
Diagramme de communication	Ce diagramme permet de dresser un bilan des interactions entre objets en numérotant les messages échangés.
Chronogramme	Ce diagramme est une alternative au diagramme de machines à états pour décrire le comportement interne d'un objet.

Figure 72. Diagrammes dynamiques de UML

Dans la modélisation de notre système nous n'utiliserons pas tous les diagrammes introduits ci-dessus, mais uniquement les diagrammes de cas d'utilisation, les diagrammes de séquences, les diagrammes de classes, les diagrammes de structure composite et les machines à états à la SDL telles que les supporte l'outil Tau G2.

IV.3.2. Modélisation et validation de l'architecture du système de navigation coopérative

Classiquement, la modélisation démarre par un diagramme de cas d'utilisation qui fait un bilan des services de base offerts par le système et des interactions avec les utilisateurs du système et le réseau.

Pour combler le fossé entre la vue fonctionnelle donnée par le diagramme de cas d'utilisation et le premier diagramme de classes qui repose par nature sur un paradigme objet, nous identifions les objets en question au travers de la construction de plusieurs diagrammes de séquences qui représentent des scénarios de cas nominaux et dégradés du comportement du système. Ces scénarios sont accompagnés d'une liste précise des hypothèses sous lesquelles la modélisation est effectuée. Ainsi, les scénarios identifiés à cette étape serviront de base à la construction incrémentale de la structure du système (diagramme de classes) par levée progressive des hypothèses restrictives et introduction des comportements correspondants.

Cette phase d'analyse est suivie d'une phase de conception qui précise le diagramme de classes éventuellement produit en phase d'analyse et décrit la structure statique du système. Les comportements des objets sont décrits par des machines à états.

Le comportement de ces objets donne au modèle un caractère exécutable permettant de mettre en œuvre une simulation avec l'outil Tau G2. Cet outil produit des traces de simulation sous forme de diagrammes de séquences que l'on peut comparer manuellement aux diagrammes de séquences définis, à titre documentaire, lors de la phase d'analyse.

IV.4. Modélisation en UML de l'architecture du système de navigation coopérative

Dans cette section nous allons présenter le processus de modélisation de l'architecture du système de navigation coopérative, selon la méthode expliquée dans le paragraphe précédent.

IV.4.1. Cas d'utilisation de CoLab

La *Figure 73* présente le diagramme de cas d'utilisation de CoLab.

Dans ce diagramme de cas d'utilisation nous pouvons observer les différents services disponibles dans CoLab. Les utilisateurs (*user*), à gauche, sont représentés en tant qu'« acteurs », conformément à la notation d'UML. Le réseau (*net*), à droite du diagramme, est également représenté en tant qu'« acteur », car c'est à partir de cette composante que les ressources demandées sont rapatriées. 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.

Nous pouvons observer comment les utilisateurs interagissent directement avec un service d'aiguillage (*DispatchingService*) et un service d'applet (*AppletService*). Le service d'aiguillage est nécessaire car plusieurs sessions peuvent s'exécuter dans un serveur, et c'est précisément ce service qui est en charge de relayer les requêtes utilisateur vers la session correspondante. Le service d'aiguillage comprend, pour chaque session existante,

un service de courtier (*BrokingService*), qui est en charge de traiter les requêtes de navigation des utilisateurs. Le service d'aiguillage peut également accéder aux services de gestion de session (*SessionManagement*). Le service de courtier, quant à lui, accède aux services de gestion de session et de gestion de la navigation (*BrowsingManagement*).

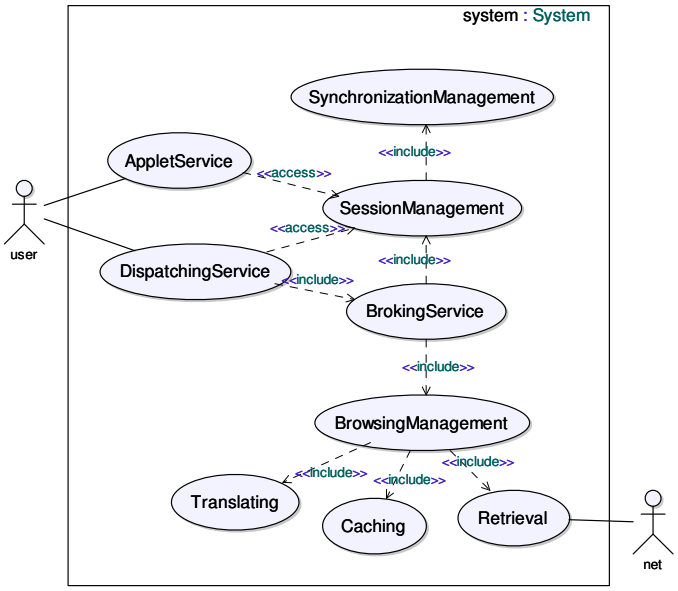


Figure 73. Cas d'utilisation de CoLab

Notons que l'utilisateur accède au service de gestion de la navigation uniquement par l'intermédiaire du courtier. Ceci est dû au fait que les requêtes de navigation ne sont pas satisfaites systématiquement, mais en fonction de certaines conditions spécifiées au niveau du gestionnaire de session.

Le gestionnaire de session comprend un service de synchronisation (*SynchronizationManagement*), qui est en charge de gérer la synchronisation des utilisateurs de la session et de garantir la cohérence de l'état de synchronisation global de la session (la notion de cohérence d'un état de synchronisation a été introduite dans le chapitre précédent).

Finalement, le service de navigation comprend trois services : un service de traduction (*Translating*), un service de cache local (*Caching*) et un service de rapatriement des ressources (*Retrieving*), qui accède aux serveurs Web distants via le réseau.

Le service d'applet (*AppletService*) correspond à l'implémentation de l'outil qui s'exécute dans le navigateur de chaque client, et qui permet à ce navigateur de communiquer avec la plate-forme de navigation coopérative. Ce service communique directement avec le service de gestion de session.

IV.4.2. Définition des scénarios d'utilisation de CoLab

Une des étapes importantes dans le processus de modélisation est de définir des scénarios d'utilisation, représentés sous la forme de diagrammes de séquences, décrivant comment le système est censé réagir à différents événements (occurrences d'action). Ces diagrammes de séquences font référence à des objets, instances des classes qui seront détaillées ultérieurement dans le paragraphe IV.4.3.

Une fois qu'une session de navigation coopérative a été configurée, elle devient disponible et un utilisateur peut exécuter plusieurs interactions avec cette session, comme :

- entrer dans une session ;
- sortir d'une session ;

- créer et supprimer des relations de synchronisation avec d'autres utilisateurs appartenant à cette session ;
- naviguer de manière synchronisée ou non sur le Web.

IV.4.2.1. Entrée dans une session de navigation coopérative

L'accès à une session de navigation coopérative est contrôlé par un mécanisme d'authentification (mot de passe). Les mots de passe ne sont pas associés ici à des utilisateurs individuels, mais aux rôles que ces utilisateurs peuvent jouer dans une session coopérative.

L'entrée d'un utilisateur dans une session de navigation coopérative est représentée par l'action « login ». Les différents scénarios possibles sont présentés dans le diagramme de séquences de la *Figure 74*.

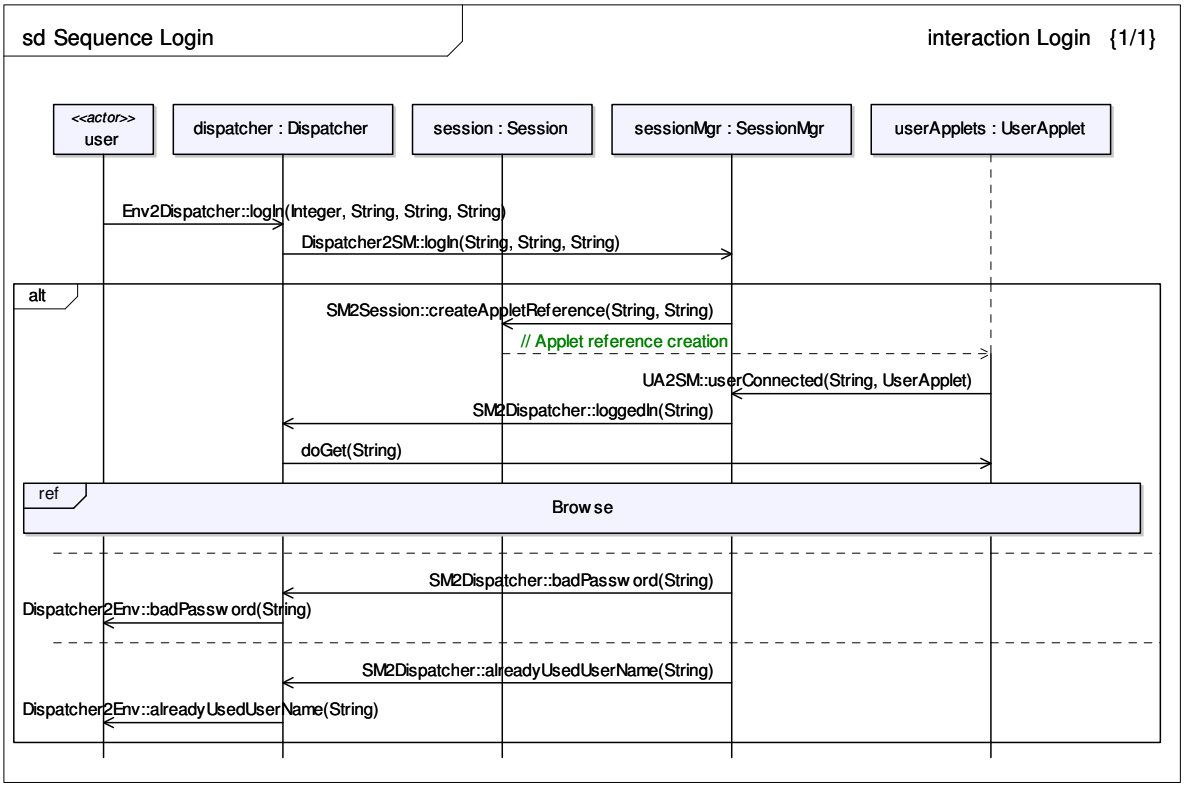


Figure 74. Diagramme de séquences de l'action « Login »

Dans ce diagramme, l'utilisateur est représenté en tant qu'acteur, comme dans le diagramme de cas d'utilisation. L'utilisateur est à l'origine de la requête d'accès à la session, qui est reçue par l'aiguilleur. Ce dernier choisit la bonne session selon la demande de l'utilisateur, et relaie la requête au gestionnaire de session. Trois scénarios sont alors possibles (les différents cas sont identifiés dans le diagramme de la *Figure 74* par les lignes pointillées).

Le premier scénario correspond à la situation où la requête de l'utilisateur peut être satisfaite. Le gestionnaire de session demande à la session de créer l'instance d'applet correspondante (cette action n'existe pas en réalité puisque l'applet de l'utilisateur est lancée automatiquement dans le navigateur, mais ceci est nécessaire dans le cadre de notre modélisation). Une fois l'applet de l'utilisateur lancée, elle envoie au gestionnaire de session un message pour lui confirmer qu'elle est en cours d'exécution. Le gestionnaire de session informe alors l'aiguilleur de cette situation, qui envoie l'URL initial de la session à

l'applet de l'utilisateur l'URL pour qu'il soit chargé. L'action de charger l'URL est représentée par la référence « Browse ».

Le second scénario correspond à la situation où l'utilisateur n'a pas fourni le bon mot de passe en fonction du rôle qu'il a choisi ; un message d'erreur est donc envoyé en réponse à la requête.

Le troisième scénario est similaire au second mais ici l'utilisateur ne peut pas s'enregistrer dans la session car un autre utilisateur a déjà choisi le même nom de login (`alreadyUserUserName`).

IV.4.2.2. Sortie d'une session de navigation coopérative

Une fois qu'un utilisateur s'est enregistré dans une session, il peut décider à tout instant d'en sortir, et ceci indépendamment de son état de synchronisation. Le diagramme de séquences associé est présenté dans la *Figure 75*.

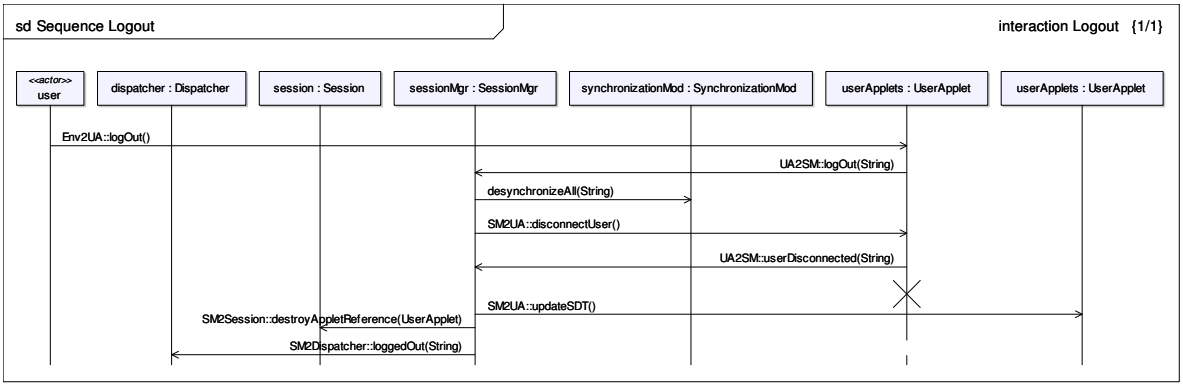


Figure 75. Diagramme de séquences de l'action « Logout »

Pour mieux illustrer ce scénario, nous avons considéré deux utilisateurs, d'où la présence de deux instances d'applet. La requête de sortie est engendrée par un utilisateur via son applet, ce qui est représenté par un message « logout » en provenance de l'environnement vers l'applet de l'utilisateur concerné. Dès la réception de ce message, l'applet envoie un message au gestionnaire de session pour l'informer de l'arrivée de cette requête, et le gestionnaire de session demande au module de synchronisation de supprimer les relations de synchronisation concernant cet utilisateur. Après divers acquittements, le gestionnaire de session envoie aux utilisateurs enregistrés dans la session une mise à jour de l'arbre de synchronisation (le SDT a été introduit dans le chapitre précédent).

IV.4.2.3. Création de relations de synchronisation

La création de relations de synchronisation entre deux utilisateurs, tel que nous l'avons expliqué dans le chapitre précédent, est exprimée par l'utilisation d'une des requêtes suivantes :

- « I_Follow_You »
- « You_Follow_Me »
- « I_Spy_You »
- « You_Join_Me »

Étant donné que les deux dernières requêtes correspondent à des spécialisations des deux premières, nous ne les considérerons pas dans ce paragraphe.

L'utilisation de la requête « I_Follow_You » peut conduire à plusieurs scénarios, qui sont présentés dans le diagramme de séquences de la *Figure 76*.

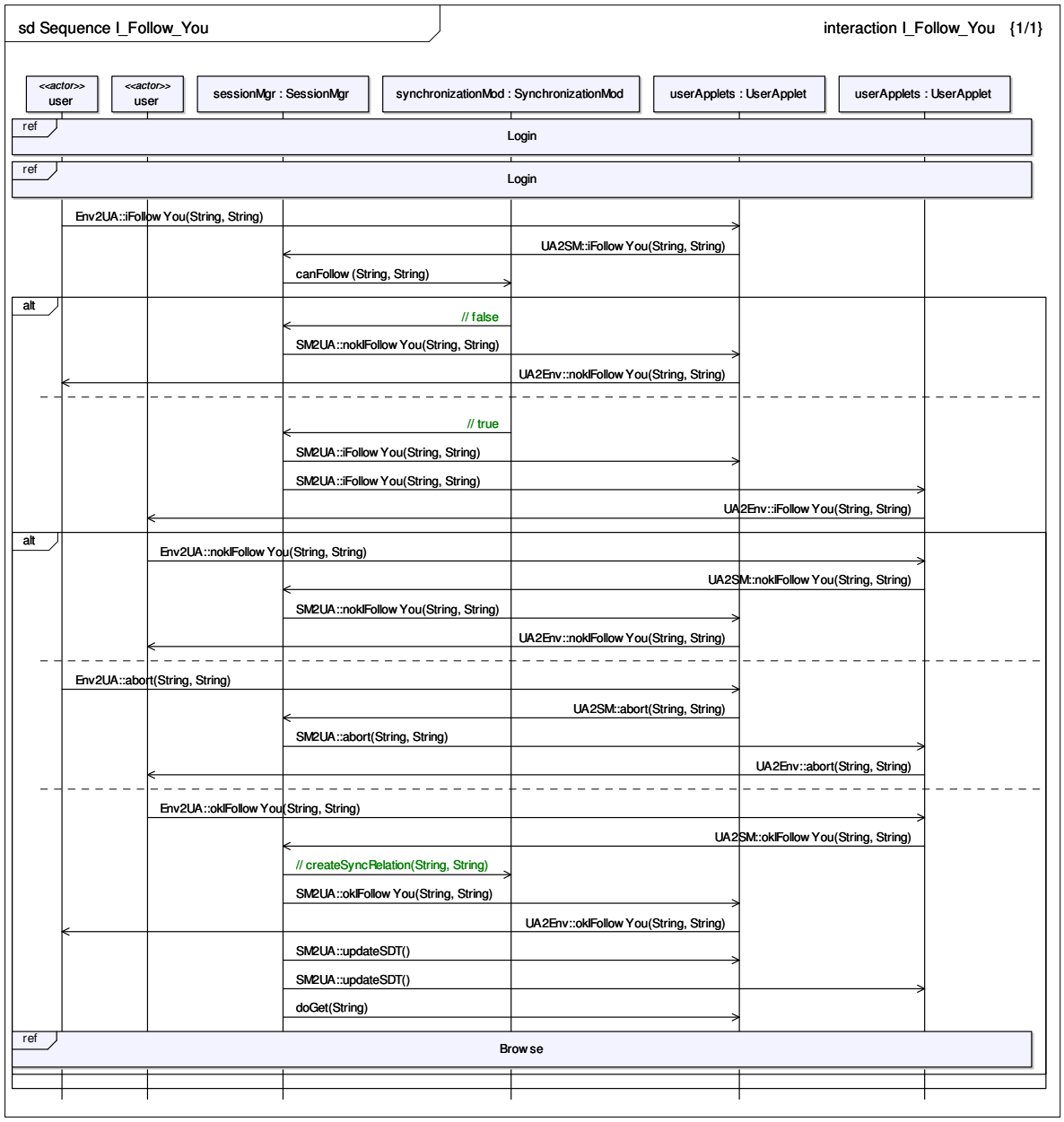


Figure 76. Diagramme de séquences de l'action « I_Follow_You »

Une condition requise pour l'application de la requête « I_Follow_You » est que les utilisateurs concernés soient déjà enregistrés dans la session, ce qui est représenté dans le diagramme de séquences par les symboles de référence « Login », un par utilisateur concerné.

Nous voyons dans ce diagramme que l'utilisateur interagit directement avec son applet, de manière similaire a ce qui a été illustré pour la requête de sortie de session. Lorsque le message est reçu par l'applet, il vérifie auprès du gestionnaire de session si la requête peut être satisfaite (en fait c'est le module de synchronisation, en tant que composant responsable de la gestion de la synchronisation qui est en charge de cette décision). Si elle ne peut pas être satisfaite, l'utilisateur est simplement informé et l'état de synchronisation de la session reste sans modification. Dans le cas contraire, le mécanisme d'autorisation est déclenché et la requête est redirigée à l'autre utilisateur. Une fois que la requête a été envoyée, trois situations peuvent se présenter :

- l'utilisateur avec lequel on veut créer la relation de synchronisation refuse l'invitation ;
- l'utilisateur qui a fait la requête décide de se rétracter ;
- l'invitation est acceptée.

Dans les deux premiers cas, le système ne fait qu'informer l'autre utilisateur de la décision prise. Dans le dernier cas, la relation de synchronisation est créée et le nouveau SDT est envoyé à tous les utilisateurs enregistrés dans la session. Le gestionnaire de session informe finalement l'applet de l'utilisateur qui vient de se synchroniser de l'URL qu'elle doit maintenant rapatrier (cette URL correspond au dernier URL chargé par l'utilisateur avec lequel on vient de se synchroniser).

Le comportement de la requête « You_Follow_Me » est similaire à celui de la requête « I_Follow_You ». Le diagramme de séquences pour cette requête est présenté dans la *Figure 77*.

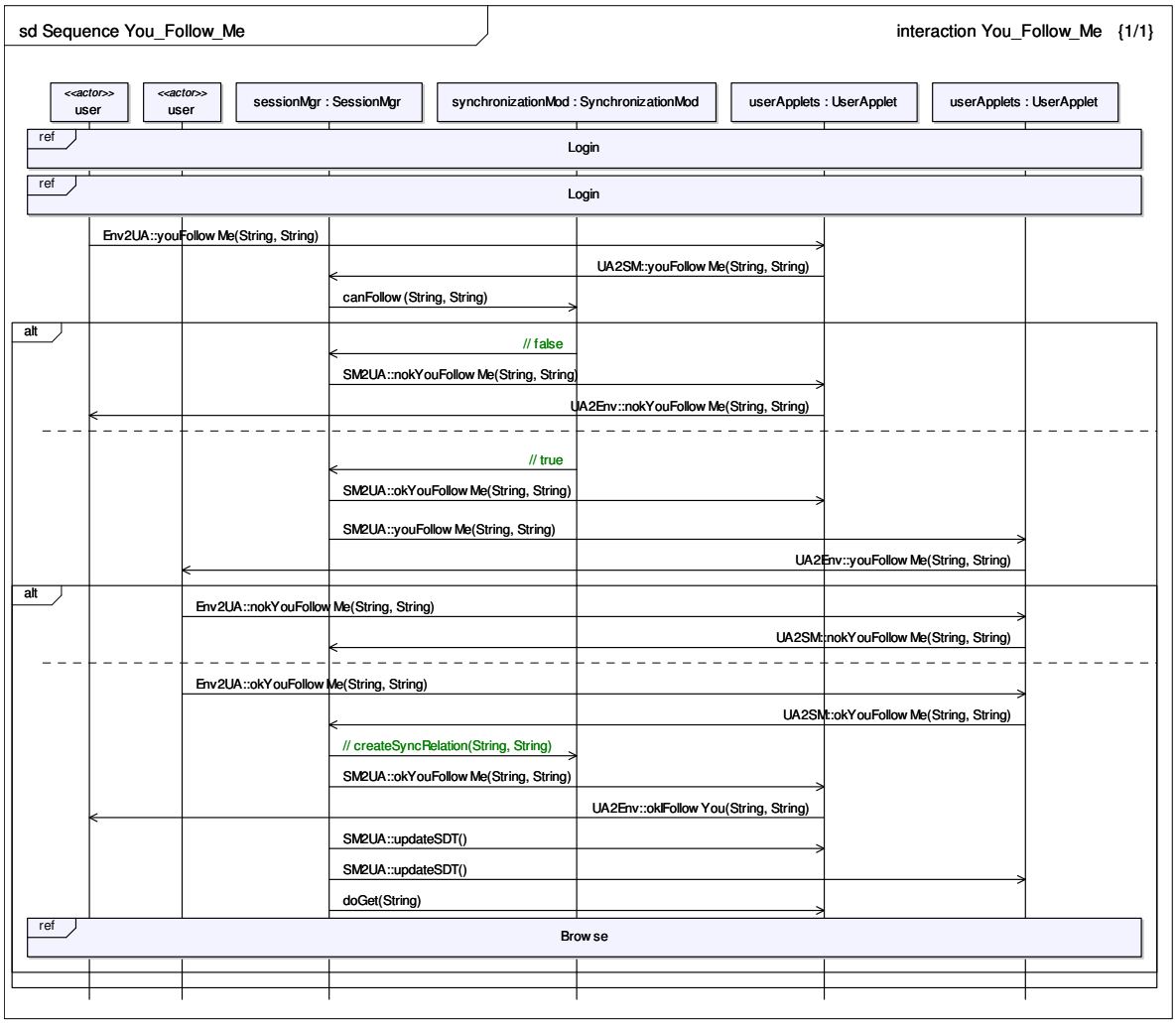


Figure 77. Diagramme de séquences de l'action « You_Follow_Me »

La différence principale entre cette requête et la requête « I_Follow_You » est que l'utilisateur qui envoie la requête ne peut pas l'annuler. Par contre, l'utilisateur auquel l'invitation est envoyée peut l'accepter ou la refuser.

IV.4.2.4. Suppression de relations de synchronisation

Une fois que des relations de synchronisation ont été créées, n'importe quel utilisateur concerné peut décider de les supprimer. Ceci est réalisé par utilisation de la requête « I_Leave_You ». L'utilisation de cette requête est inconditionnelle et son diagramme de séquences est présenté dans la *Figure 78*.

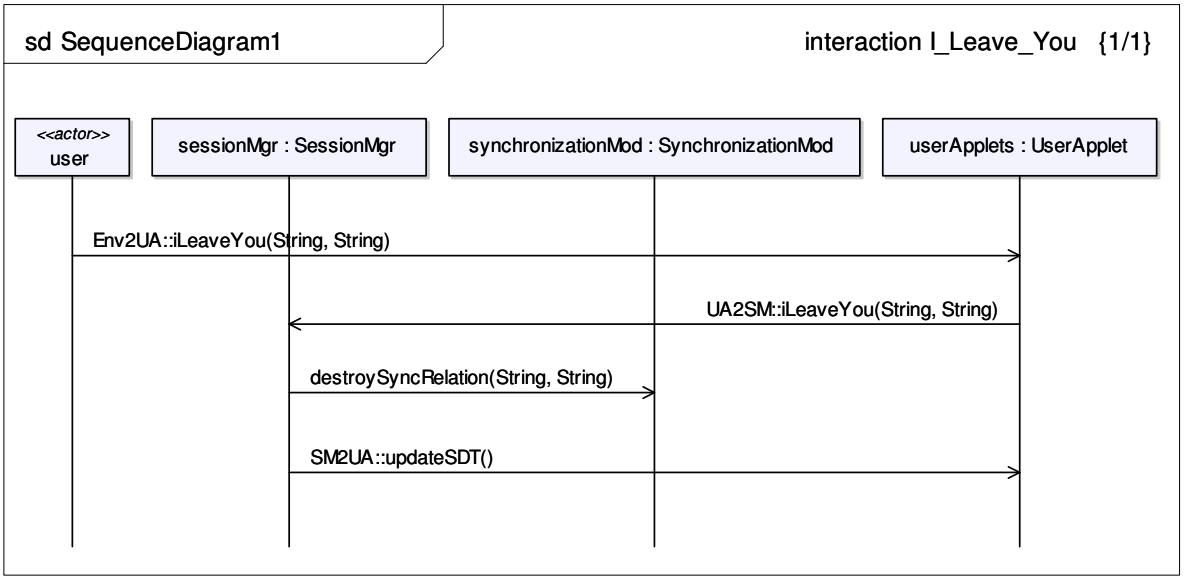


Figure 78. Diagramme de séquences de l'action « I_Leave_You »

Lorsque l'utilisateur exprime, à travers son applet, son intention d'éliminer la relation de synchronisation, la requête est satisfaite, et le nouveau SDT est envoyé aux applets de tous les utilisateurs connectés à la session, pour que l'affichage soit actualisé dans leurs navigateurs.

IV.4.2.5. Navigation sur le Web

Le diagramme de séquences pour l'action de navigation est présenté dans la *Figure 79*.

Nous pouvons voir qu'une requête de navigation peut être engendrée soit directement par l'utilisateur (par exemple, en cliquant sur un hyperlien ou en entrant un URL dans la barre du navigateur), soit par le mécanisme de synchronisation de la navigation lorsqu'un l'utilisateur est synchronisé à un autre utilisateur (à travers son applet). Indépendamment par qui elle a été engendrée, elle arrive directement à l'aiguilleur.

Lorsqu'une requête de navigation arrive au courtier, il vérifie auprès du gestionnaire de session si l'utilisateur est autorisé à rapatrier la ressource spécifiée. S'il n'est pas autorisé (par exemple, il essaie de rapatrier une ressource différente de celle actuellement affichée par l'utilisateur avec lequel il est synchronisé), il reçoit comme réponse un message d'erreur. Dans le cas contraire, la requête est envoyée au gestionnaire de navigation pour qu'elle soit traitée.

Le gestionnaire de navigation vérifie si la ressource demandée dans la requête est déjà stockée dans le système de cache local (*cacheMod*). Dans l'affirmative, la ressource est rapatriée directement de celui-ci, et la réponse est envoyée à l'utilisateur qui a engendré la requête de navigation. Dans la négative, le gestionnaire de navigation demande au module de rapatriement de rapatrier la ressource. Une fois la ressource rapatriée, le gestionnaire de navigation demande au module de traduction de traduire cette ressource, et la ressource traduite est envoyée à la fois au module de cache local (la ressource traduite

est ainsi disponible pour d'ultérieures requêtes) et au gestionnaire de navigation pour que celui-ci la relaie à l'utilisateur via le courtier.

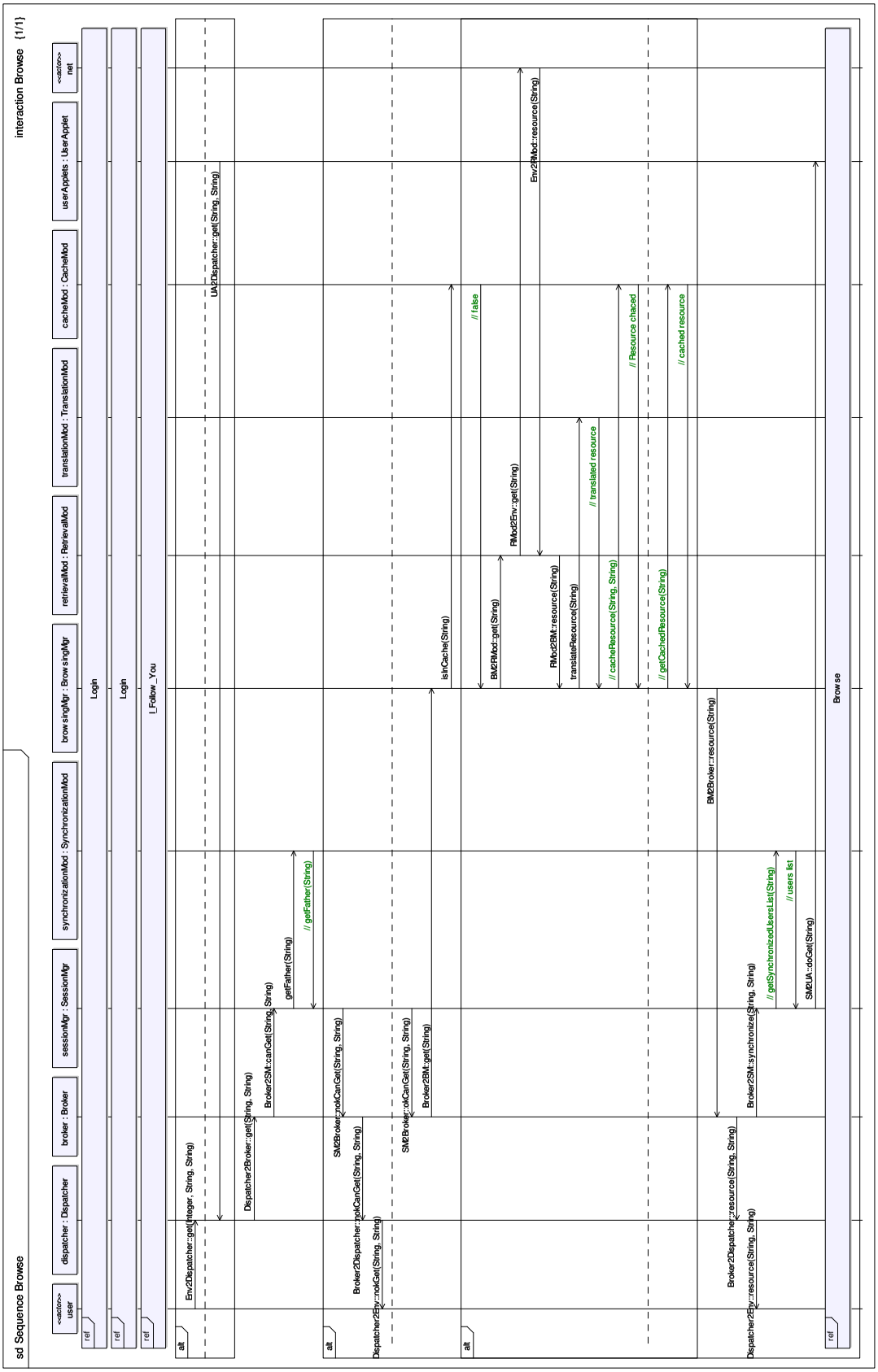


Figure 79. Diagramme de séquences de l'action « Browse »

Finalement le courtier demande au gestionnaire de session de synchroniser la navigation de tous les utilisateurs qui sont actuellement synchronisés avec l'utilisateur qui vient d'exécuter la requête de navigation. Le gestionnaire de session rapatrie alors la liste des utilisateurs synchronisés du module de synchronisation, et demande à l'applet de chacun de ces utilisateurs de rapatrier l'URL en question.

IV.4.3. Diagramme de classes de CoLab

Le diagramme de classes de la *Figure 80* décrit la structure statique du système CoLab.

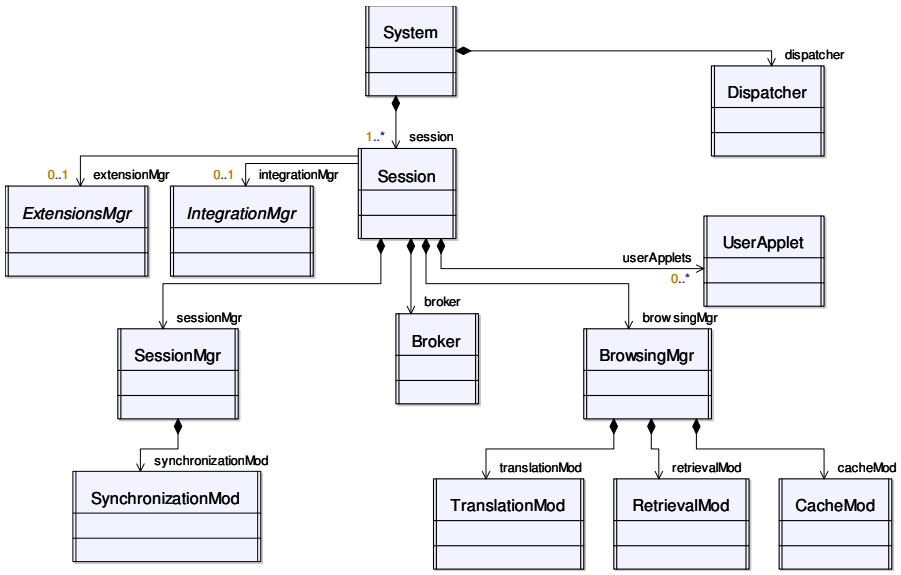


Figure 80. Diagramme de classes de CoLab

La classe de plus haut niveau, appelée *System*, est composée de plusieurs classes : l'aiguilleur (*Dispatcher*) et une ou plusieurs sessions (*Session*). La classe *Session* est quant à elle composée de quatre classes qui sont :

- *SessionMgr* qui définit le gestionnaire de session ;
- *Broker* qui définit le courtier ;
- *BrowsingMgr* qui définit le gestionnaire de navigation ;
- *UserApplet* qui définit l'applet présente dans le navigateur de chaque utilisateur.

Les deux classes définies comme ayant des relations d'association avec la classe *Session*, à savoir *ExtensionsMgr* et *IntegrationMgr*, correspondent à des modules qui permettront dans le futur :

- d'ajouter des fonctionnalités à CoLab, tel qu'un module de contrôle d'accès aux ressources [Wang-99] [Duflos-02] [Shen-92] [Damianou-01], ou d'un module permettant d'adapter la présentation des ressources rapatriées en fonction de différentes conditions, tel que le rôle de l'utilisateur, ou le type d'équipement qu'il utilise [Han-00] ; et
- d'intégrer CoLab à d'autres applications coopératives.

Notons finalement que la classe *SessionMgr* a une relation de composition avec la classe *SynchronizationMod* (module de synchronisation des utilisateurs) et que la classe *BrowsingMgr* a des relations de composition avec trois autres classes :

- *TranslationMod* qui correspond au module de traduction des ressources rapatriées ;
- *RetrievalMod* qui correspond au module de rapatriement des ressources ;
- *CacheMod* qui correspond au système de cache local des ressources rapatriées.

IV.4.3.1. La classe System

La classe System correspond à la classe de plus haut niveau de la hiérarchie des classes de notre architecture. Sa définition est présentée dans la *Figure 81*.

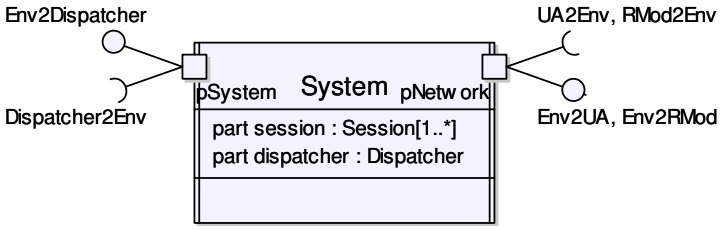


Figure 81. Définition de la classe System

La *Figure 81* définit les ports de la classe *System*, les entrées-sorties associées à ces ports et les attributs de cette classe. Sur la *Figure 82* le diagramme de structure composite montre que, conformément au diagramme de classes de la *Figure 80,* les classes *Dispatcher* et *Session* sont définies comme « parties » de la classe *System*.

Cette classe a deux ports associés : `pSystem` et `pNetwork`. Le port `pSystem` reçoit en entrée les messages définis par l'interface `Env2Dispatcher`, et envoie en sortie les messages définis par l'interface `Dispatcher2Env`.

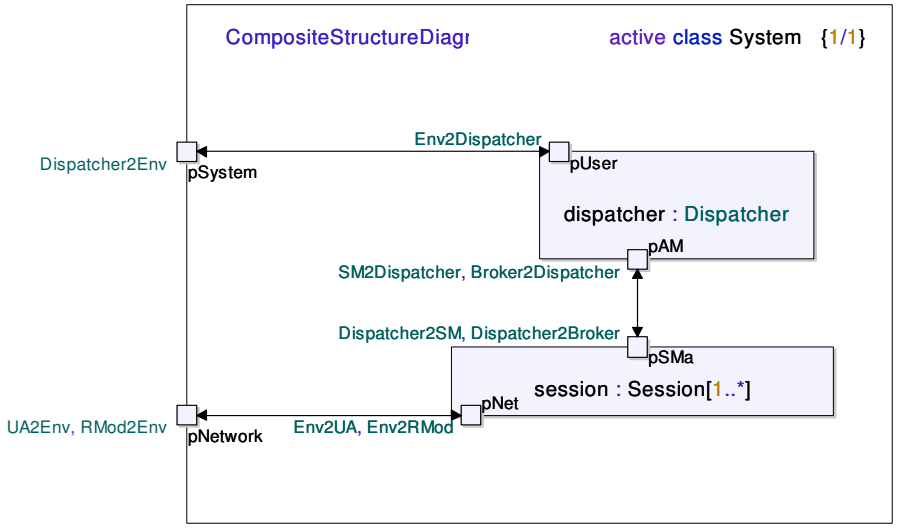


Figure 82. Diagramme de structure composite de la classe System

Le schéma de connexion entre les parties représente la façon dont elles communiquent. Ainsi, par exemple, la partie `dispatcher` a un port, `pUser`, qui est connecté au port `pSystem` de la classe System, ce qui lui permet d'envoyer et de recevoir des messages vers/de l'extérieur. Un canal de communication est défini entre ces deux ports. L'interface `Env2Dispatcher` définit l'ensemble des messages qui peuvent être reçus par le port `pUser`, et l'interface `Dispatcher2Env` définit l'ensemble des messages qui peuvent être émis par ce même port.

De manière analogue, il existe deux autres canaux, l'un qui établit une connexion entre les ports `pAM` (Dispatcher) et `pSMa` (Session), ce qui permet l'échange de messages entre l'aiguilleur et les sessions, et l'autre qui établit une connexion entre les ports `pNet` (Session) et `pNetwork` (System), ce qui permet aux sessions d'envoyer et de recevoir des messages vers/de l'extérieur.

IV.4.3.2. La classe Dispatcher

Cette classe (*Figure 83*) correspond à l'implantation d'un service de routage des requêtes des utilisateurs vers la session dans laquelle ils se sont enregistrés.

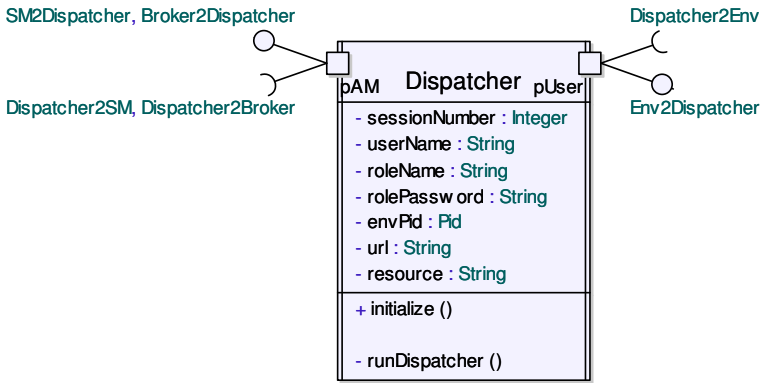
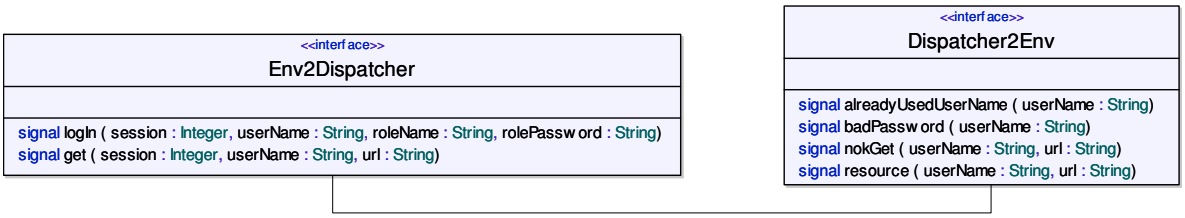


Figure 83. Définition de la classe Dispatcher

La *Figure 83* met en évidence les ports de la classe mais aussi ses attributs et méthodes. Les attributs et méthodes peuvent être définis comme publics (précédés par un caractère '+') ou privés (précédés par un caractère '-').

Sans la *Figure 84* nous présentons es interfaces `Env2Dispatcher` et `Dispatcher2Env`. Elles caractérisent les actions réalisées par les utilisateurs de la session, y compris les requêtes de rapatriement des ressources Web.



Interface Env2Dispatcher

Message	Signification
`login(sessionNumber, userName, roleName, password)`	Demande d'accès d'un utilisateur à la session *sessionNumber*, en utilisant le nom d'utilisateur *userName*, et en jouant le rôle *roleName* avec le mot de passe *password*.
`get(sessionNumber, userName, url)`	Requête de rapatriement de la ressource identifiée par url par l'utilisateur *userName* dans la session *sessionNumber*.

Interface Dispatcher2Env

Message	Signification
`alreadyUsedUsername(userName)`	Indique à l'utilisateur que le nom d'utilisateur qu'il a choisi a déjà été pris par autre utilisateur.
`badPassword(userName)`	Indique à l'utilisateur que le mot de passe qu'il a tapé ne correspond pas au mot de passe du rôle choisi.
`nokGet(userName, url)`	Informe l'utilisateur qu'il n'a pas le droit de rapatrier la ressource demandée.
`resource(userName, url)`	Envoi de la ressource à l'utilisateur en réponse à sa requête.

Figure 84. Interfaces de la classe Dispatcher

IV.4.3.3. La classe Session

La classe Session regroupe l'ensemble des composants responsables du traitement d'une session de navigation coopérative.

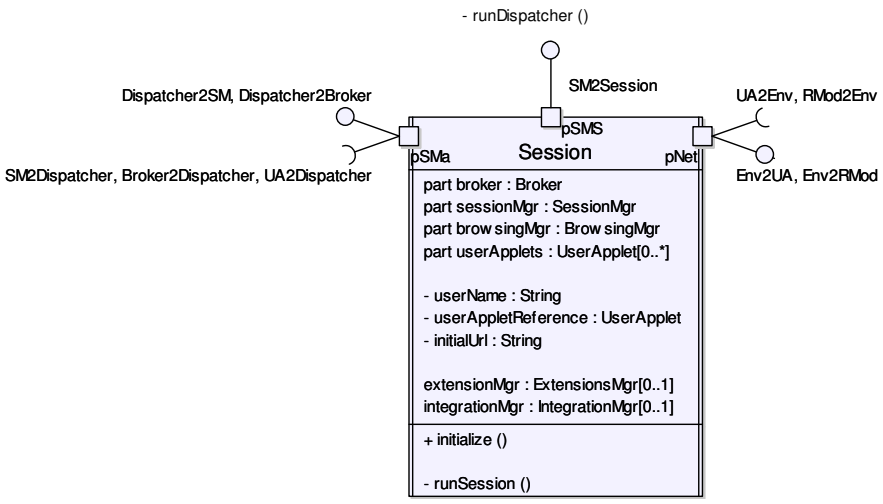


Figure 85. Définition de la classe Session

Dans la *Figure 85,* nous pouvons voir la combinaison des trois types d'éléments que nous avons vu dans les définitions de classes précédentes : les parties, les attributs et les méthodes. Dans le cas de ce diagramme nous pouvons voir aussi que la structure proposée dans la *Figure 80* est respectée, et que les classes Broker, SessionMgr, BrowsingMgr et UserApplet sont considérées comme étant « parties » de cette classe.

Dans la *Figure 86* nous présentons le diagramme de structure composite de cette classe.

Une session est constituée d'un gestionnaire de session (`sessionMgr : SessionMgr`), d'un courtier (`broker : Broker`), d'un gestionnaire de navigation (`browsingMgr : BrowsingMgr`) et de zéro ou plusieurs applets utilisateurs (`userApplet : UserApplet[0..*]`). Elle peut également être constituée de deux éléments optionnels additionnels dont nous avons parlé précédemment, à savoir un gestionnaire d'extensions (`extensionMgr : ExtensionMgr[0..1]`) et un gestionnaire d'intégration (`integrationMgr : IntegrationMgr[0..1]`). Ces éléments ne sont présents ici qu'à titre informationnel et leur modélisation ne sera pas plus détaillée.

La *Figure 86* montre le schéma d'interconnexion des ports entre composants définis dans cette classe. Le courtier, à travers le port `pEnv`, qui est connecté au port `pSMa`, peut envoyer les messages définis dans l'interface `Broker2Dispatcher`. En faisant référence au diagramme de structure composite de la *Figure 82*, nous pouvons remarquer que tous ces messages ont comme destinataire final l'aiguilleur. Les deux autres ports du courtier lui permettent de communiquer avec le gestionnaire de session (`pSM`) et avec le gestionnaire de navigation (`pBM`), ce qui correspond à l'explication intuitive que nous avons présentée dans la *Figure 69*.

Une remarque importante est qu'un même port peut avoir des connexions avec plusieurs autres ports. Le routage des messages est alors réalisé en fonction des ensembles de signaux (interfaces) définis pour chacun de ports en fonction des ports avec lesquels il existe une connexion (canal).

En ce qui concerne les communications de cette classe, nous constatons qu'il n'y a qu'une seule interface définie, qui sert au gestionnaire de session à envoyer à la session les ordres de créer ou de supprimer des références à des instances d'applets des utilisateurs. Cette interface est présentée dans la *Figure 87*.

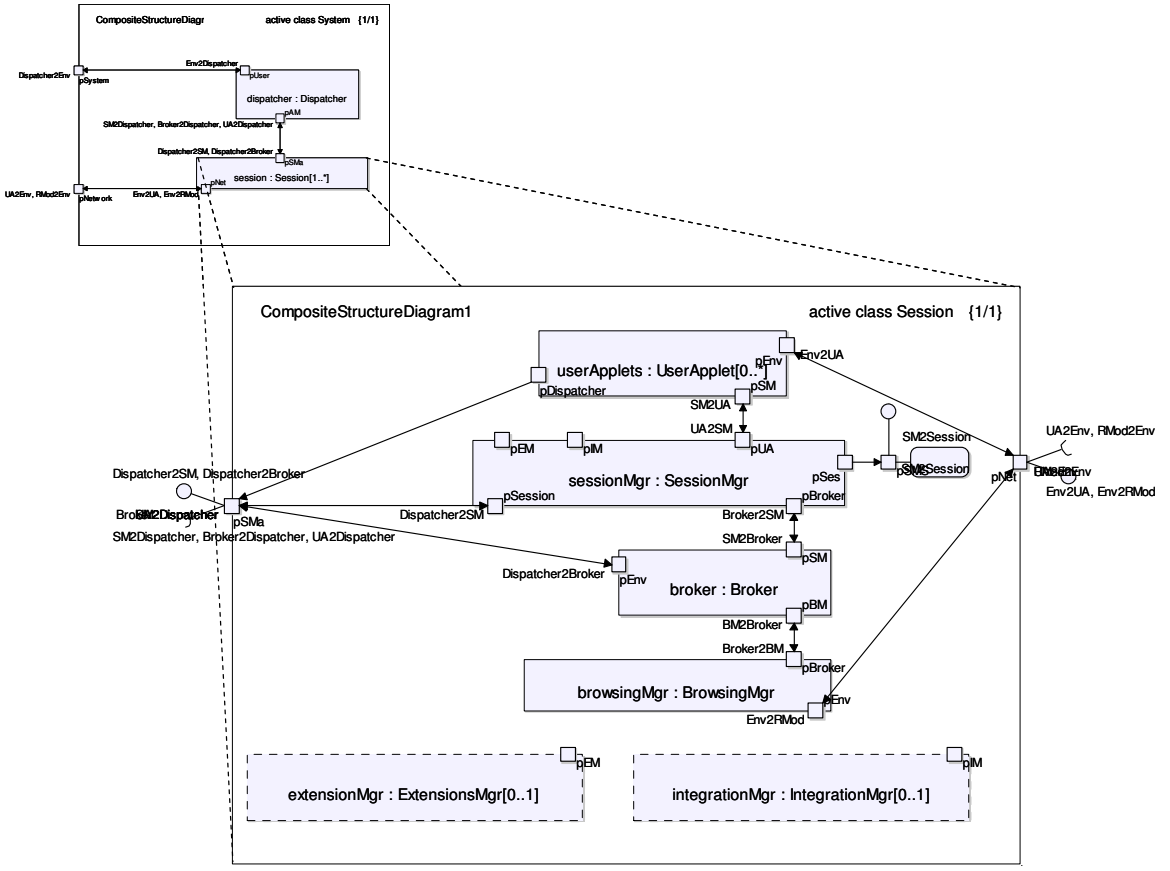
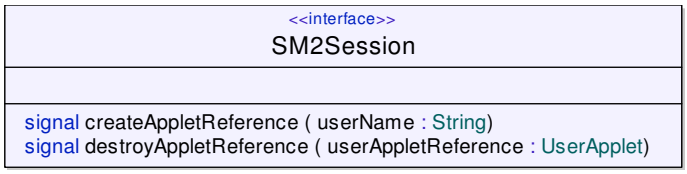


Figure 86. Diagramme de structure composite de la classe Session



Interface SM2Session

Message	Signification
`createAppletReference(userName)`	Demande à l'entité session de créer une nouvelle instance de la classe *UserApplet* pour l'utilisateur indiqué.
`destroyAppletReference(userApplet)`	Demande à l'entité session de détruire la référence de l'applet indiquée.

Figure 87. Interface de la classe Session

IV.4.3.4. La classe Broker

Le courtier a le rôle d'intermédiaire entre les requêtes des utilisateurs et le gestionnaire de navigation. Pour chaque requête, le courtier demande au gestionnaire de session si elle peut être satisfaite (i.e. si, par exemple, l'utilisateur a le droit de rapatrier une ressource). Si la réponse est affirmative, le courtier demande au gestionnaire de navigation de rapatrier la ressource, et une fois le rapatriement réalisé, il lui demande d'envoyer la réponse à l'utilisateur concerné. Dans le cas contraire, il envoie un message d'erreur à

l'utilisateur. La définition de cette classe est présentée dans la *Figure 88*, et ses interfaces dans la *Figure 89*.

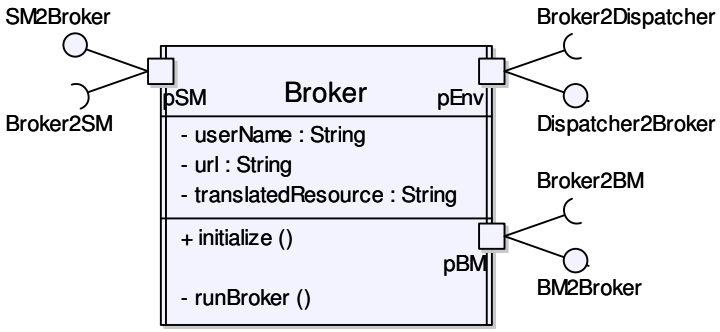
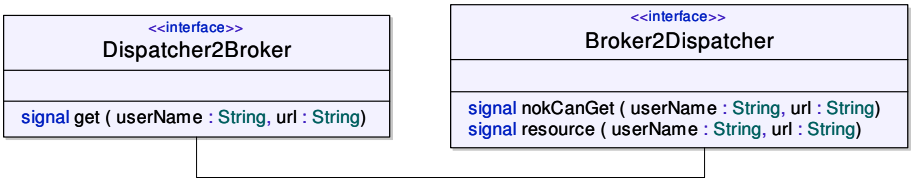


Figure 88. Définition de la classe Broker



Interface Dispatcher2Broker

Message	Signification
`get(userName, url)`	Requête de rapatriement d'une ressource pour un utilisateur.

Interface Broker2Dispatcher

Message	Signification
`nokCanGet(userName, url)`	Indique à l'aiguilleur que la requête de rapatriement de la ressource ne peut être satisfaite pour cet utilisateur.
`resource(userName, url)`	Envoi de la ressource à l'aiguilleur en réponse à la requête de l'utilisateur.

Figure 89. Interfaces de la classe Broker

IV.4.3.5. La classe SessionMgr

Cette classe prend en charge la gestion de la session de navigation coopérative. Sa principale responsabilité est de gérer la session dans son ensemble et, grâce au module de synchronisation (`SynchronizationMod`), les requêtes de synchronisation des utilisateurs. Elle a également pour objectif de garantir la cohérence globale de l'état de synchronisation. La définition de la classe est présentée dans la *Figure 90*, et ses interfaces dans la *Figure 91*.

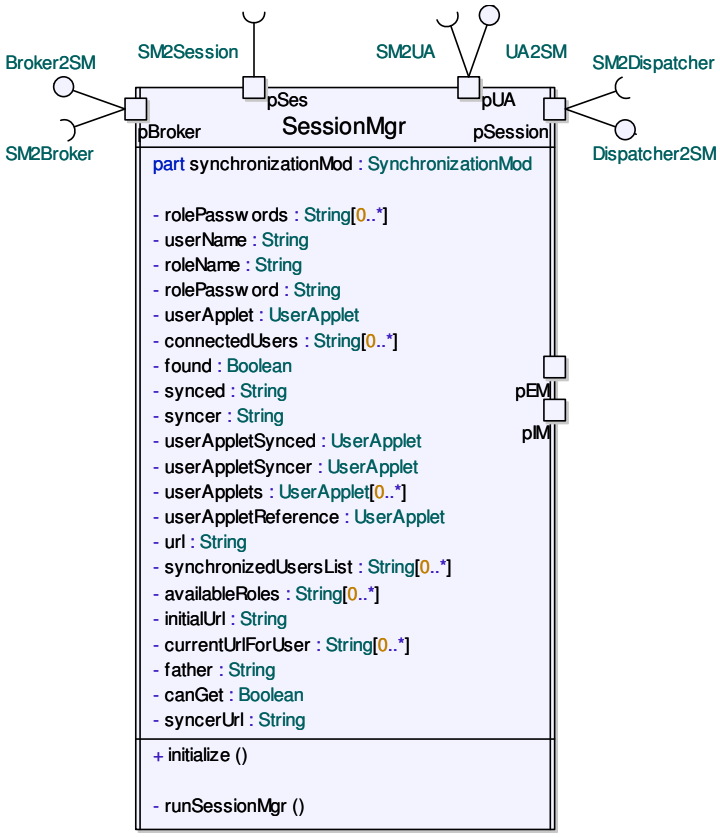
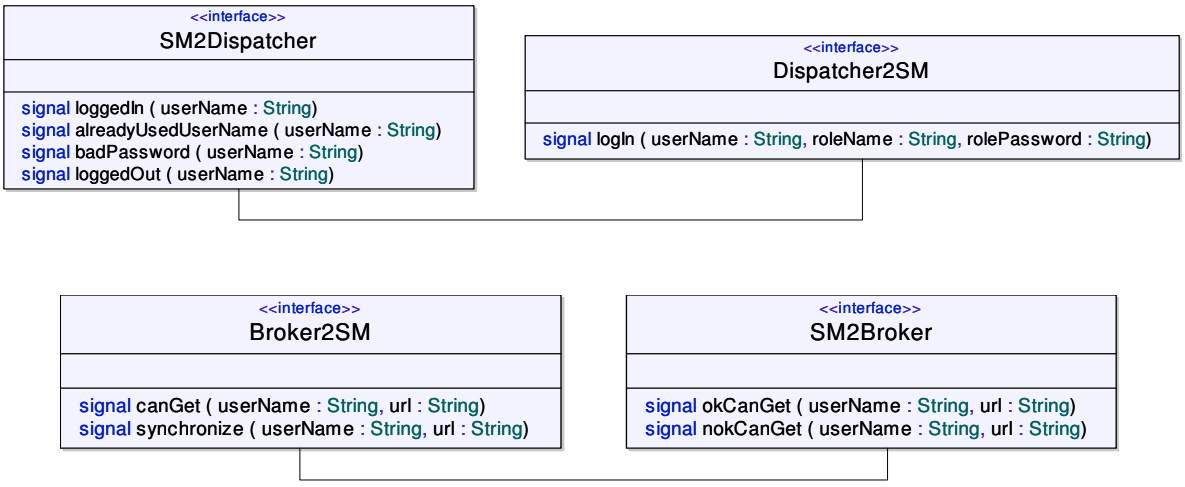


Figure 90. Définition de la classe SessionMgr



Interface SM2Dispatcher

Message	Action
`loggedIn(userName)`	Le gestionnaire de session informe l'aiguilleur que la requête d'entrée dans la session pour l'utilisateur indiqué a abouti.
`alreadyUsedUsername(userName)`	Le gestionnaire de session informe l'aiguilleur que le nom d'utilisateur choisi par l'utilisateur a déjà été pris par un autre utilisateur.
`badPassword(userName)`	Le gestionnaire de session informe l'aiguilleur que le mot de passe tapé par l'utilisateur ne correspond pas au mot de passe du rôle choisi.
`loggedOut(userName)`	Le gestionnaire de session informe l'aiguilleur que l'utilisateur indiqué est sorti de la session.

Interface Dispatcher2SM

Message	Action
`logIn(userName, roleName, password)`	L'aiguilleur informe le gestionnaire de session de la présence d'une requête d'entrée dans la session de la part d'un utilisateur.

Interface Broker2SM

Message	Action
`canGet(userName, url)`	Le courtier demande au gestionnaire de session si l'utilisateur a le droit de rapatrier la ressource spécifiée.
`synchronize(userName, url)`	Le courtier demande au gestionnaire de session de synchroniser la navigation web pour tous les utilisateurs actuellement synchronisés avec l'utilisateur spécifié.

Interface SM2Broker

Message	Signification
`okCanGet(userName, url)`	Le gestionnaire de session informe le courtier que l'utilisateur a le droit de rapatrier la ressource demandée.
`nokCanGet(userName, url)`	Le gestionnaire de session informe le courtier que l'utilisateur n'a pas le droit de rapatrier la ressource demandée.

Figure 91. Interfaces de la classe SessionMgr

IV.4.3.6. La classe SynchronizationMod

Cette classe concerne l'implémentation du mécanisme de synchronisation des utilisateurs. Elle comprend l'ensemble des fonctionnalités nécessaires pour garantir qu'en présence de requêtes de création de relations de synchronisation, il n'est pas possible d'engendrer des états incohérents (cette notion de cohérence a été introduite dans le chapitre précédent).

Dans la *Figure 92*, nous présentons la définition de cette classe. Nous n'avons pas d'interfaces dans ce cas, puisque les méthodes publiques proposées par ce module sont appelées directement par le gestionnaire de session.

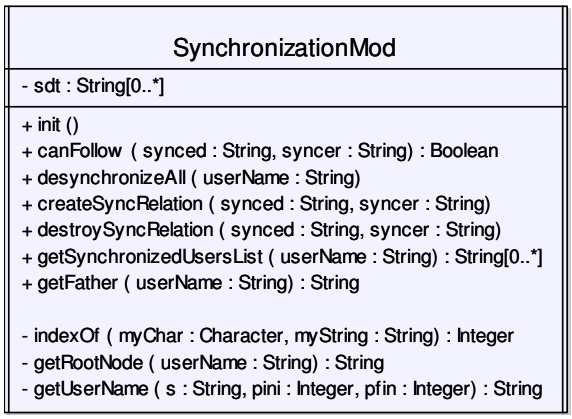


Figure 92. Définition de la classe SynchronisationMod

IV.4.3.7. La classe BrowsingMgr

Cette classe représente la partie du système en charge de rapatrier les ressources demandées par les utilisateurs. La définition de cette classe est présentée dans la *Figure 93*.

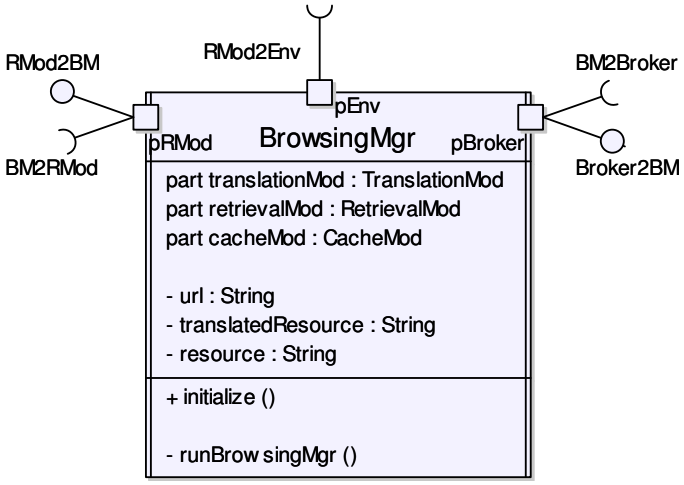


Figure 93. Définition de la classe BrowsingMgr

Dans ce diagramme nous retrouvons la structure définie dans la *Figure 80*. Nous voyons ainsi qu'il existe des instances `translationMod : TranslationMod`, `retrievalMod : RetrievalMod` et `cacheMod : CacheMod`, qui font partie intégrale de cette classe. Le diagramme de structure composite correspondant est présenté dans la *Figure 94*.

Dans ce diagramme la seule instance à avoir des ports est `retrievalMod`, qui communique avec l'extérieur par le port `pEnv` qui permet de rapatrier des ressources Web distantes. Le port `pBroker` permet à cette classe d'échanger des messages avec le courtier. Le port `pRMod` permet au gestionnaire de session d'échanger des messages avec le module de rapatriement. Les deux autres instances, `cacheMod` et `translationMod`, contiennent des méthodes publiques qui sont directement accessibles par le gestionnaire de navigation. La description des interfaces associées à cette classe sont présentées dans la *Figure 95*.

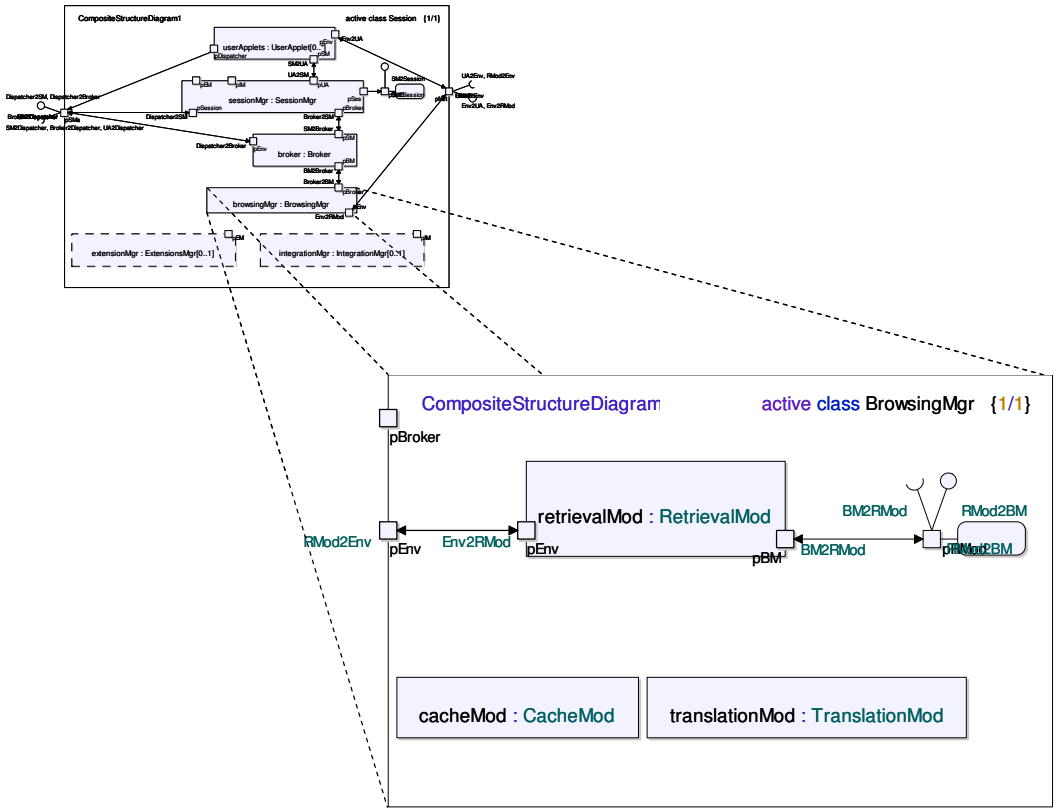
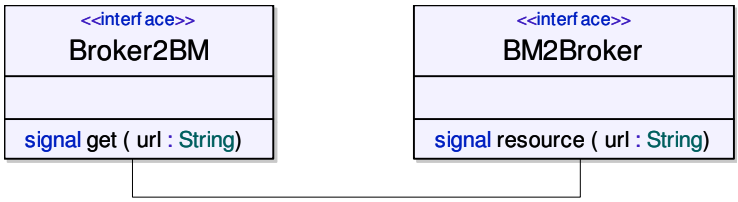


Figure 94. Diagramme de structure composite de la classe BrowsingMgr



Interface Broker2BM

Message	Signification
`get(userName, url)`	Le courtier demande au gestionnaire de navigation de rapatrier la ressource pour l'utilisateur.

Interface BM2Broker

Message	Signification
`resource(userName, url)`	Envoi au courtier de la ressource demandée.

Figure 95. Interfaces de la classe BrowsingMgr

IV.4.3.8. Les modules RetrievalMod, CacheMod et de TranslationMod

Ces trois composants constituent le noyau du gestionnaire de navigation.

Le module de rapatriement est responsable du rapatriement de ressources Web à partir de serveurs distants raccordés à un réseau informatique. Il met donc en œuvre tout le logiciel nécessaire au traitement des requêtes exprimées en utilisant le protocole HTTP.

Le module de traduction modifie les ressources HTML rapatriées pour qu'elles puissent être traités par CoLab (cet aspect sera détaillé dans le chapitre suivant).

Le module de cache local permet de mettre en cache les ressources rapatriées. Les ressources préalablement stockées dans ce module seront utilisées par des utilisateurs (utilisateurs dits synchrones dans le chapitre précédent) dont la navigation est synchronisée à celle d'un autre utilisateur (utilisateur dit asynchrone dans le chapitre précédent) et qui est associé à la racine d'un arbre de synchronisation.

Les classes de ces modules sont présentées dans la *Figure 96*, et leurs interfaces sont décrites dans la *Figure 97*.

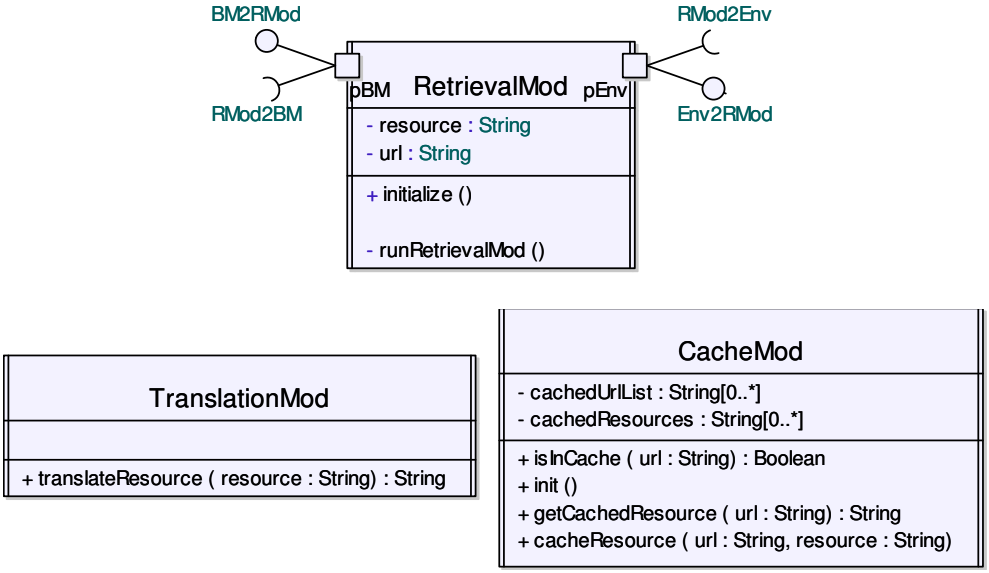
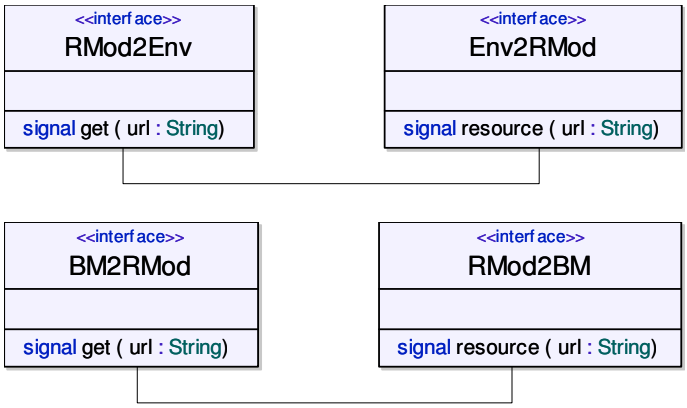


Figure 96. Les classes RetrievalMod, TranslationMod et CacheMod



Interface RMod2Env

Message	Signification
`get(url)`	Requête de rapatriement d'une ressource.

Interface Env2RMod

Message	Signification
`resource(url)`	Rapatriement de la ressource demandée.

Interface BM2RMod

Message	Signification
`get(url)`	Requête de rapatriement d'une ressource.

Message	Signification
`resource(url)`	Envoi de la ressource demandée.

Figure 97. Interfaces de la classe RetrievalMod

IV.4.3.9. La classe UserApplet

Cette classe représente la partie du système qui réside dans le navigateur de chaque utilisateur. À travers une instance de cette classe, l'utilisateur peut demander la création et la suppression de relations de synchronisation avec d'autres utilisateurs. Cette classe sert également à recevoir les URLs qui devront être chargés dans le navigateur de l'utilisateur, lorsque sa navigation Web sera synchronisée à celle d'un autre utilisateur. Dans la *Figure 98* nous présentons la définition de cette classe, et dans la *Figure 99* les interfaces associées.

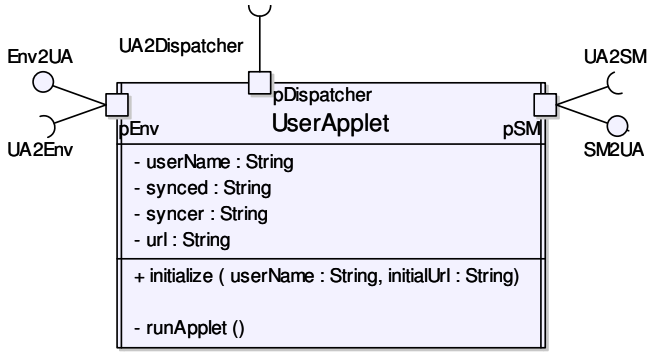
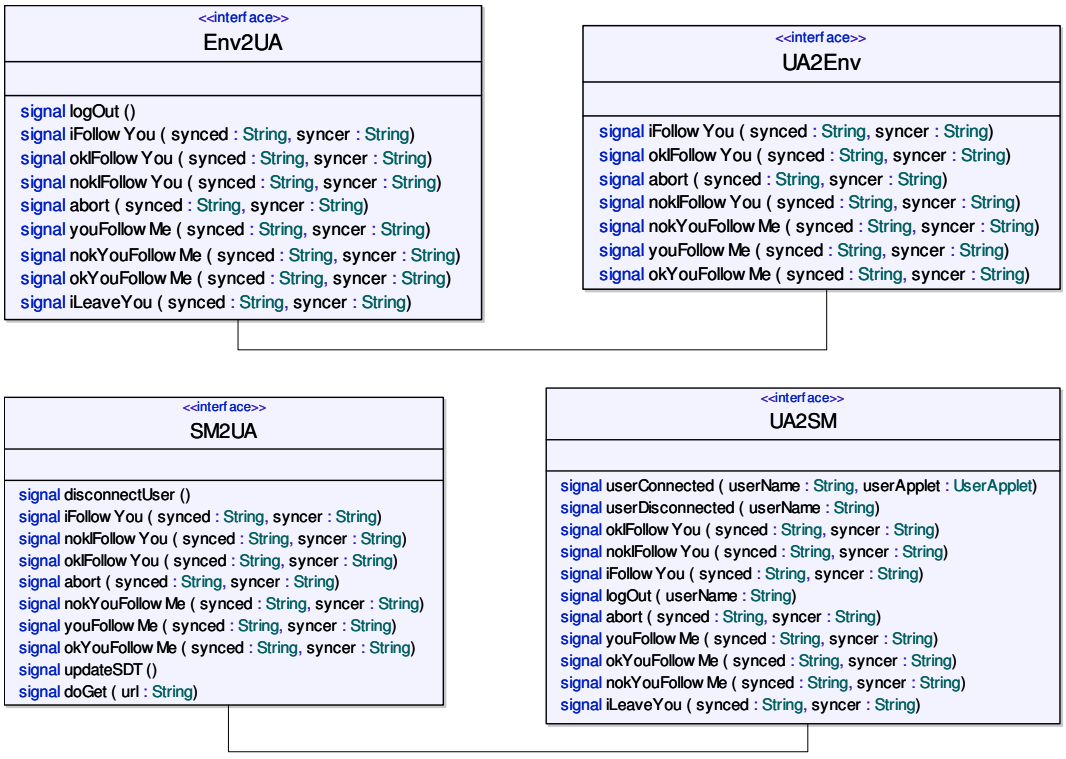
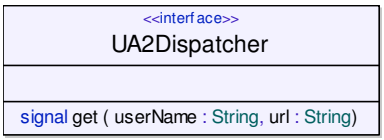


Figure 98. La classe UserApplet





Interface Env2UA & SM2UA

Message	Signification
`logout()`	Demande de sortie d'une session.
`iFollowYou(synced, syncer)`	Demande de création d'une relation de synchronisation de type « I_Follow_You » de l'utilisateur « synced » à l'utilisateur « syncer ».
`okIFollowYou(synced, syncer)`	Acceptation de création d'une relation de synchronisation de type « I_Follow_You ».
`nokIFollowYou(synced, syncer)`	Refus de création d'une relation de synchronisation de type « I_Follow_You ».
`abort(synced, syncer)`	Annulation d'une requête de création d'une relation de synchronisation de type « I_Follow_You ».
`youFollowMe(synced, syncer)`	Demande de création d'une relation de synchronisation de type « You_Follow_Me » de l'utilisateur « syncer » à l'utilisateur « synced ».
`okYouFollowMe(synced, syncer)`	Acceptation de création d'une relation de synchronisation de type « You_Follow_Me ».
`nokYouFollowMe(synced, syncer)`	Refus de création d'une relation de synchronisation de type « You_Follow_Me ».
`iLeaveYou(synced, syncer)`	Suppression d'une relation de synchronisation entre les utilisateurs « synced » et « syncer ».
`disconnectUser()`	Demande à l'applet de l'utilisateur de se déconnecter de la session, suite à une requête de l'utilisateur de sortir de la session.
`updateSDT()`	Envoi d'une nouvelle structure de synchronisation suite à une modification des relations de synchronisation.
`goGet(url)`	Demande à l'applet de l'utilisateur de rapatrier une ressource.

Interface UA2Dispatcher

Message	Signification
`get(userName, url)`	Requête de rapatriement d'une ressource.

Interface UA2Env & UA2SM

Message	Signification
`iFollowYou(synced, syncer)`	Demande de création d'une relation de synchronisation de type « I_Follow_You » de l'utilisateur « synced » à l'utilisateur « syncer » envoyée à l'utilisateur « syncer », en réponse à sa requête.
`okIFollowYou(synced, syncer)`	Acceptation de création d'une relation de synchronisation de type « I_Follow_You » envoyée à l'utilisateur « synced » en réponse à sa requête.
`nokIFollowYou(synced, syncer)`	Refus de création d'une relation de synchronisation de type « I_Follow_You » envoyée à l'utilisteur « synced » en réponse à sa requête.
`abort(synced, syncer)`	Annulation d'une requête de création d'une relation de synchronisation de type « I_Follow_You » envoyée à l'utilisateur « syncer » en réponse à sa requête.
`youFollowMe(synced, syncer)`	Demande de création d'une relation de synchronisation de type « You_Follow_Me » de l'utilisateur « syncer » à l'utilisateur « synced » envoyée à l'utilisateur « syncer ».
`okYouFollowMe(synced, syncer)`	Acceptation de création d'une relation de synchronisation de type « You_Follow_Me » envoyée à l'utilisateur « syncer » en réponse à sa requête.
`nokYouFollowMe(synced, syncer)`	Refus de création d'une relation de synchronisation de type « You_Follow_Me » envoyée à l'utilisateur « syncer » en réponse à sa requête.
`userConnected(userName)`	Indication au gestionnaire de session que l'utilisateur s'est enregistré dans la session.
`userDisconnected(userName)`	Indication au gestionnaire de session que l'utilisateur s'est déconnecté de la session.

Figure 99. Interfaces de la classe UserApplet

IV.5. Validation de la modélisation de l'architecture

Dans ce paragraphe nous allons présenter les diagrammes de séquences produits par des simulations des cas les plus représentatifs de notre modèle et nous verrons qu'ils correspondent bien aux scénarios d'utilisation de CoLab que nous avons présentés dans le paragraphe IV.4.2.

IV.5.1. Machines à états décrivant le comportement des classes

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. À partir de la définition de ces comportements, nous avons engendré des diagrammes de séquence décrivant les interactions entre les différents composants de notre système. Afin de ne pas surcharger le mémoire, nous ne détaillerons les diagrammes SDL que nous avons développés que pour trois requêtes de CoLab à savoir « Login », « I_Follow_You » et « Browse », qui peuvent être considérées comme trois des requêtes les plus représentatives de notre modèle.

IV.5.1.1. L'action « Login »

Nous présentons dans la *Figure 100* le comportement de la classe aiguilleur en cas d'arrivée d'une requête d'entrée dans une session.

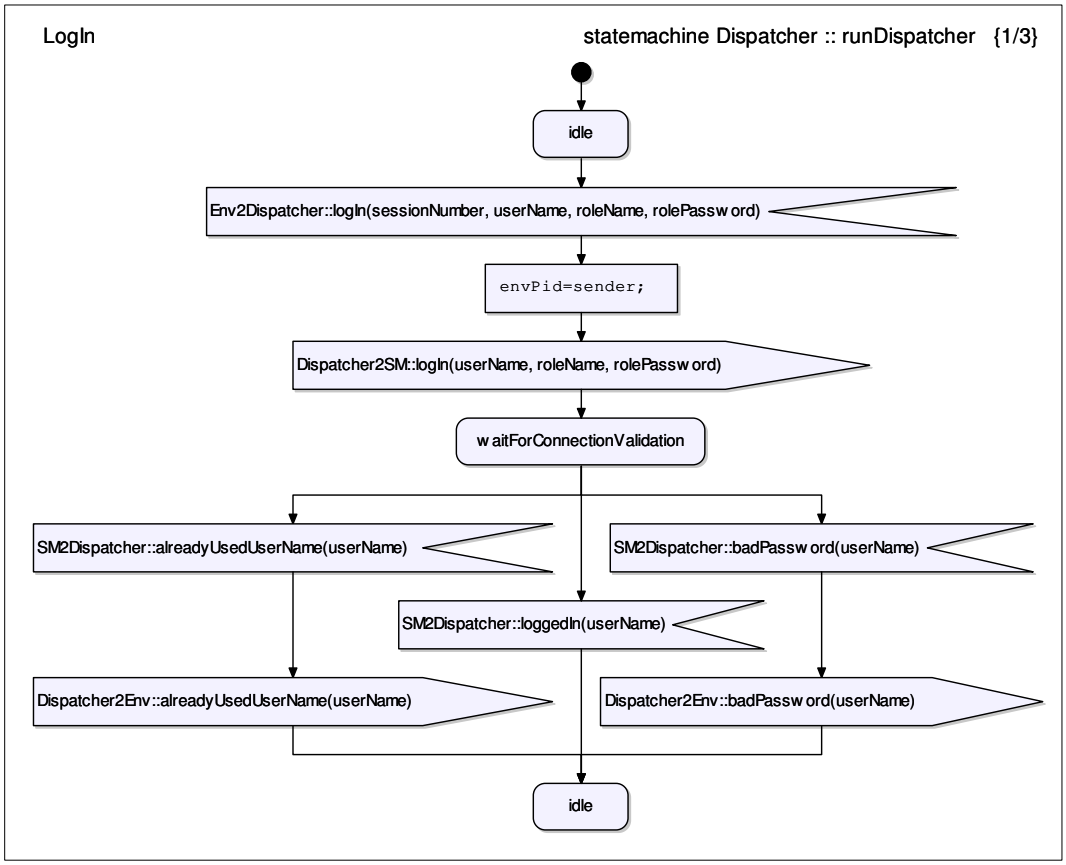


Figure 100. Diagramme SDL de l'action « Login » pour la classe aiguilleur

Ainsi, en présence du message (appelé signal dans le profil SDL de Tau G2) `Env2Dispatcher::logIn(sessionNumber,userName,roleName,rolePassword)`, provenant de l'environnement (c'est à dire d'un utilisateur), l'aiguilleur exécute une action interne et envoie au gestionnaire de session le message `Dispatcher2SM::logIn(userName,roleName,rolePassword)`. Il reste ensuite dans un état intermédiaire, dans l'attente d'une réponse de ce dernier. Trois réponses sont possibles : `SM2Dispatcher::alreadyUsedUserName(userName)`, `SM2Dispatcher::badPassword(userName)`, ou `SM2Dispatcher::loggedIn(userName)`. En fonction de la réponse obtenue, l'aiguilleur envoie ou pas des messages à l'environnement.

Le diagramme SDL du gestionnaire de session pour le traitement de cette même action est présenté dans la *Figure 101*.

Lors de la réception du message `Dispatcher2SM::logIn(userName,roleName, rolePassword)`, le gestionnaire de session vérifie si le nom de rôle et le mot de passe sont corrects. Si ce n'est pas le cas, il envoie en réponse à l'aiguilleur le message `SM2Dispatcher::badPassword(userName)`. Dans le cas contraire, le gestionnaire de session vérifie s'il existe déjà un utilisateur enregistré dans la session qui utilise le nom d'utilisateur proposé dans le message reçu. Si c'est le cas, il envoie à l'aiguilleur le message `SM2Dispatcher::alreadyUsedUserName(userName)`. Dans le cas contraire, il demande la création de l'instance de l'applet de l'utilisateur (message `SM2Session::createAppletReference(userName,initialUrl)`), et se met en attente

de confirmation (message `UA2SM::userConnected(userName,userApplet)`). Une fois ce message reçu, le gestionnaire de session enregistre l'information nécessaire, et envoie le message `SM2Dispatcher::loggedIn(userName)` à l'aiguilleur, et le message `SM2UA::doGet(initialUrl)` à l'instance de l'applet qui vient d'être créée pour demander le chargement de l'URL initial de la session dans le navigateur de l'utilisateur.

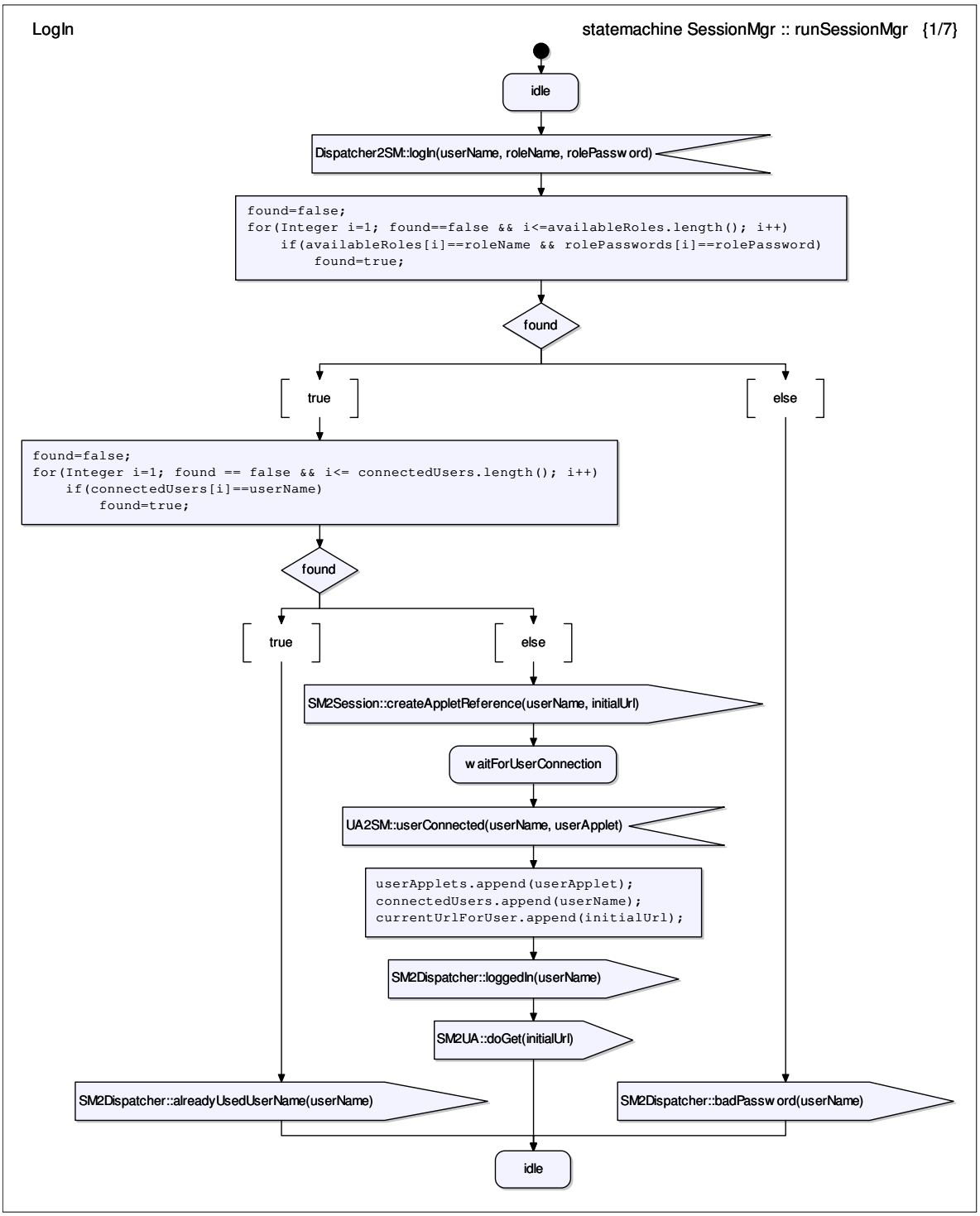


Figure 101. Diagramme SDL de l'action « Login » pour la classe gestionnaire de session

De son côté, l'instance de la classe UserApplet, au moment qu'elle est créée, elle envoie le message `UA2SM::userConnected(userName,userApplet)` pour informer qu'il

a correctement démarré. Le diagramme SDL de cette opération de la part de l'applet est présenté dans la .

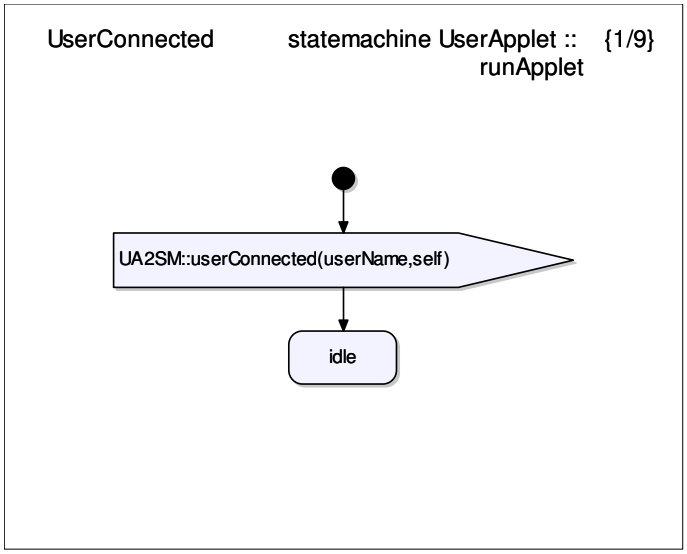


Figure 102. Diagramme SDL confirmant le démarrage de l'applet de l'utilisateur

IV.5.1.2. L'action « I_Follow_You »

Cette requête est engendrée par l'utilisateur, et envoyée directement à l'applet de son navigateur. Le diagramme SDL de l'applet de l'utilisateur qui engendre cette requête de synchronisation est présenté dans la *Figure 103*.

Lors de l'arrivée du message `Env2UA::iFollowYou(synced,syncer)`, l'applet envoie le message `UA2SM::iFollowYou(synced,syncer)` au gestionnaire de session, dans le but que ce dernier décide si la requête de l'utilisateur peut être satisfaite ou pas. Deux réponses sont possibles : `SM2UA::nokIFollowYou(synced,syncer)` ou `SM2UA::iFollowYou(synced,syncer)`. La première indique qu'il n'est pas possible de donner suite à la requête de synchronisation à cause du risque de création d'un état de synchronisation incohérent. La deuxième informe l'applet du cas contraire et la requête de synchronisation peut être alors annulée (message `Env2UA::abort(synced,syncer)`), refusée (message `SM2UA::nokIFollowYou(synced,syncer)`) ou acceptée (message `SM2UA::okIFollowYou(synced,syncer)`).

Le diagramme SDL de l'applet de l'utilisateur qui reçoit cette requête est présenté dans la *Figure 104*.

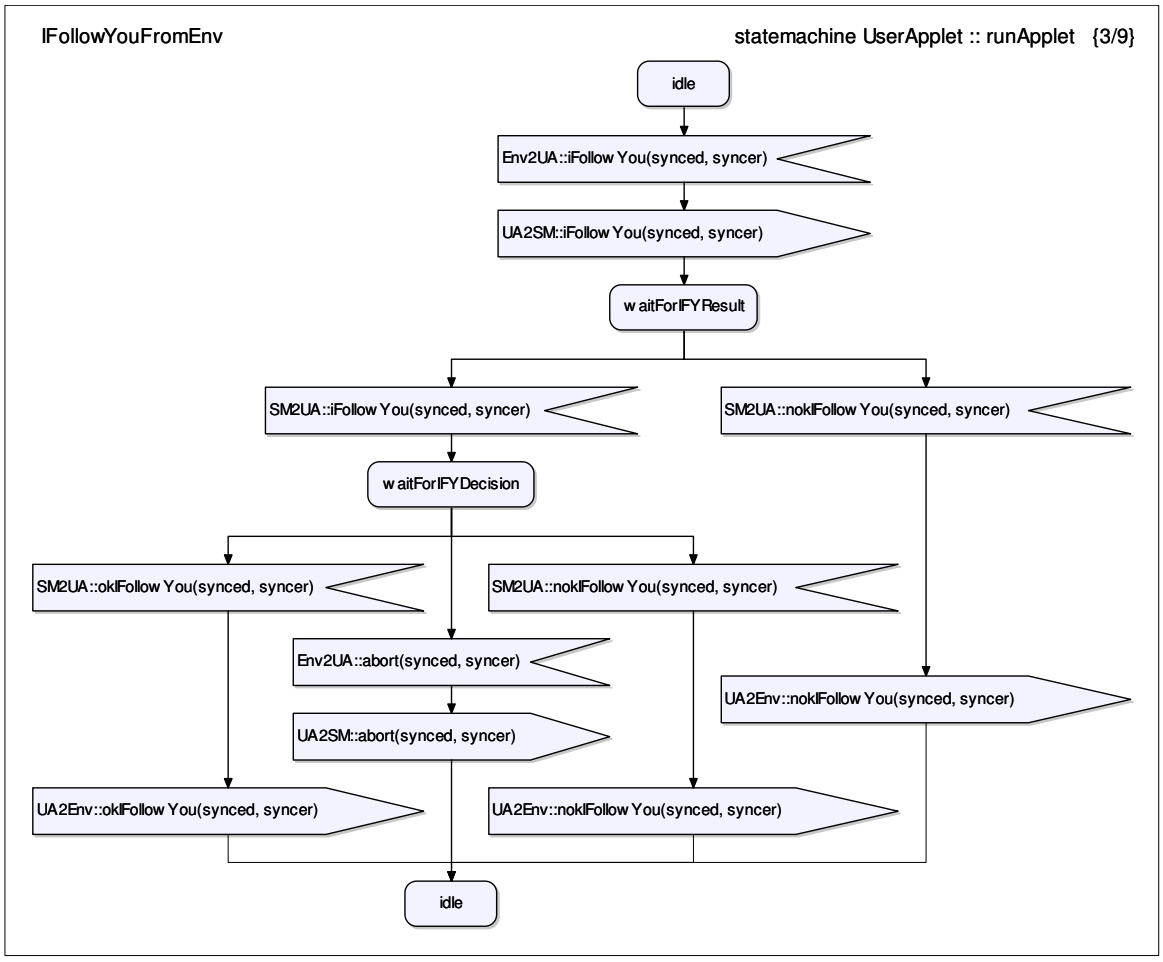


Figure 103. Diagramme SDL de l'action « I_Follow_You » pour la classe applet de l'utilisateur (celui qui envoie la requête)

Lorsque l'applet de l'utilisateur reçoit l'invitation de synchronisation (message `SM2UA::iFollowYou(synced,syncer)`), elle envoie un message à l'utilisateur pour lui en faire part (message `UA2Env::iFollowYou(synced,syncer)`), et reste dans l'attente de la réponse (messages `Env2UA::okIFollowYou(synced,syncer)`, `Env2UA::nokIFollowYou(synced,syncer)` ou `SM2UA::abort(synced,syncer)`).

L'autre composant qui intervient activement dans la réalisation de cette requête est le gestionnaire de session, dont le comportement est décrit par le diagramme SDL de la *Figure 105*.

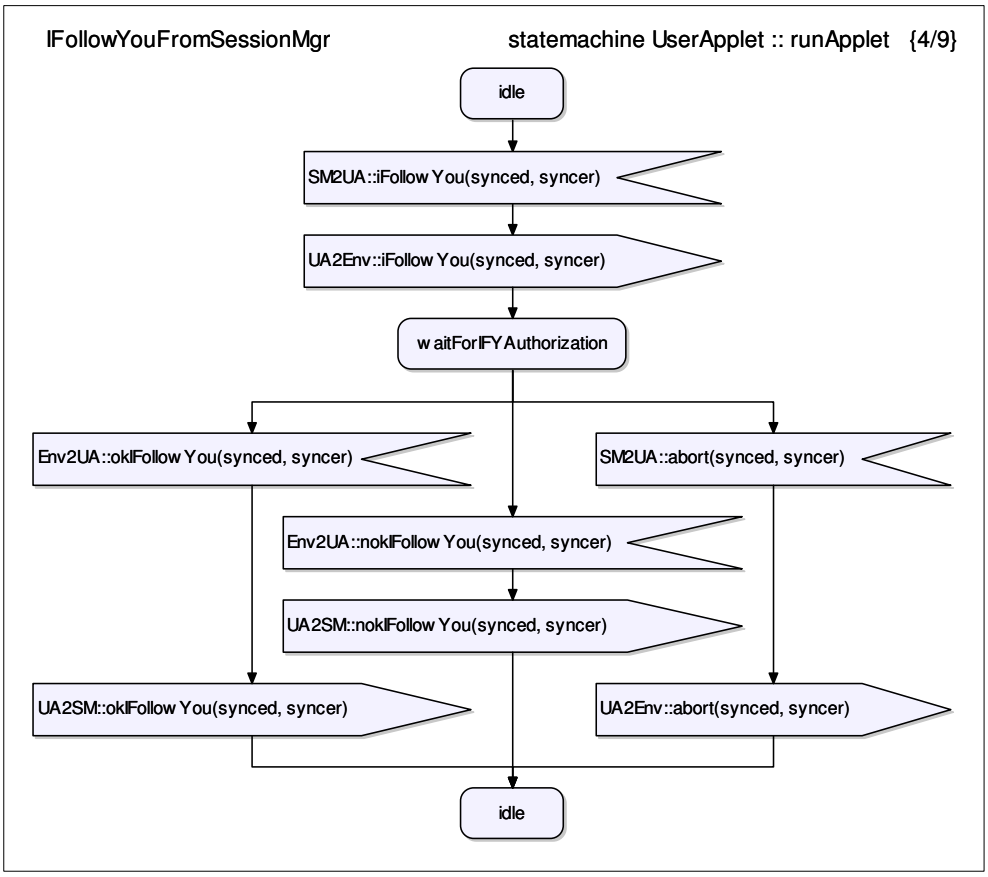


Figure 104. Diagramme SDL de l'action « I_Follow_You » pour la classe applet de l'utilisateur (celui qui reçoit la requête)

Lors de l'arrivée du message `UA2SM::iFollowYou(synced,syncer)`, le gestionnaire de session demande au module de synchronisation s'il est possible de faire suivre cette requête. Dans la négative, il envoie simplement le message `userAppletSynced.SM2UA::nokIFollowYou(synced,syncer)` à l'applet de l'utilisateur qui a engendré la requête. Autrement il envoie aux applets des deux utilisateurs concernés le message `userAppletSynce(r|d).SM2UA::iFollowYou(synced,syncer)`, et il reste dans un état d'attente. Dans cet état, trois réponses peuvent arriver : `UA2SM::okIFollowYou(synced,syncer)`, `UA2SM::nokIFollowYou(synced,syncer)` ou `UA2SM::abort(synced,syncer)`. Dans le premier cas, le gestionnaire de session demande au module de synchronisation de créer la relation de synchronisation, de notifier à l'applet de l'utilisateur que la requête de synchronisation a abouti. Il lui envoie également l'URL qui devra être chargée dans le navigateur ; cet URL correspond à l'URL actuellement chargé dans le navigateur de l'utilisateur avec lequel on vient de se synchroniser. Les deux autres cas correspondent aux situations où la relation de synchronisation a été refusée ou annulée, et le gestionnaire de session ne fait qu'envoyer les messages nécessaires aux acteurs concernés.

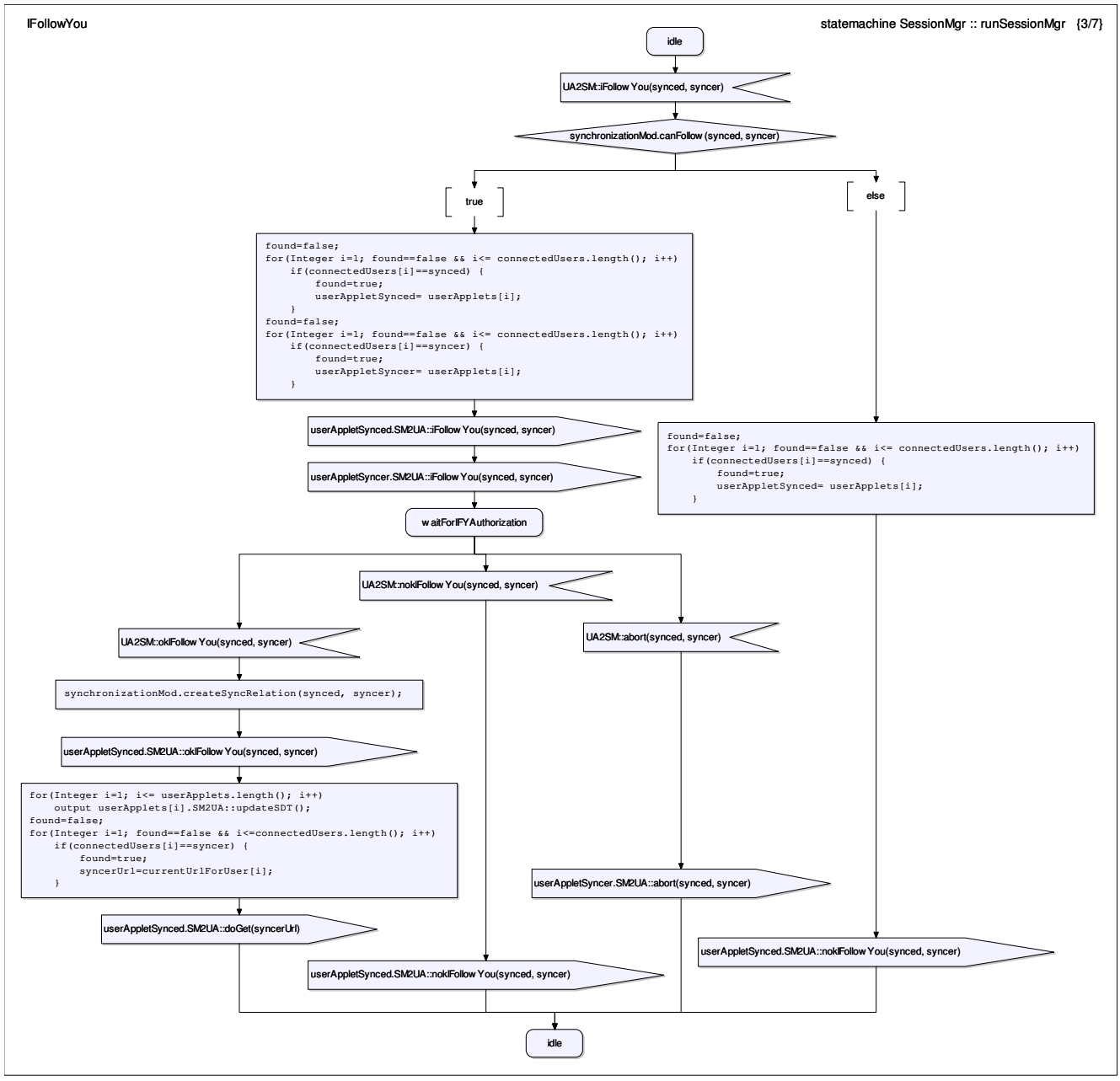


Figure 105. Diagramme SDL de l'action « I_Follow_You » pour la classe gestionnaire de session

IV.5.1.3. L'action « Browse »

Le composant de notre modèle qui reçoit les requêtes de navigation des utilisateurs est l'aiguilleur. Nous décrivons ainsi, dans un premier temps, le comportement de l'aiguilleur en présence d'une requête de navigation par le diagramme SDL de la *Figure 106*.

Le message de navigation peut arriver soit de l'environnement (i.e. une requête de navigation d'un utilisateur : `Env2Dispatcher::get(sessionNumber,userName,url)`), ou de l'applet d'un utilisateur (i.e. conséquence d'une synchronisation de la navigation : `UA2Dispatcher::get(userName,url)`). Dans les deux cas l'aiguilleur relaie la requête au courtier (message `Dispatcher2Broker::get(userName,url)`), et reste dans l'attente d'une réponse, qui peut arriver sous la forme de la ressource demandée (message `Broker2Dispatcher::resource(userName,resource)`), ou sous la forme d'un message

indiquant que la requête n'a pu être satisfaite (message `Broker2Dispatcher::nokCanGet(userName,url)`).

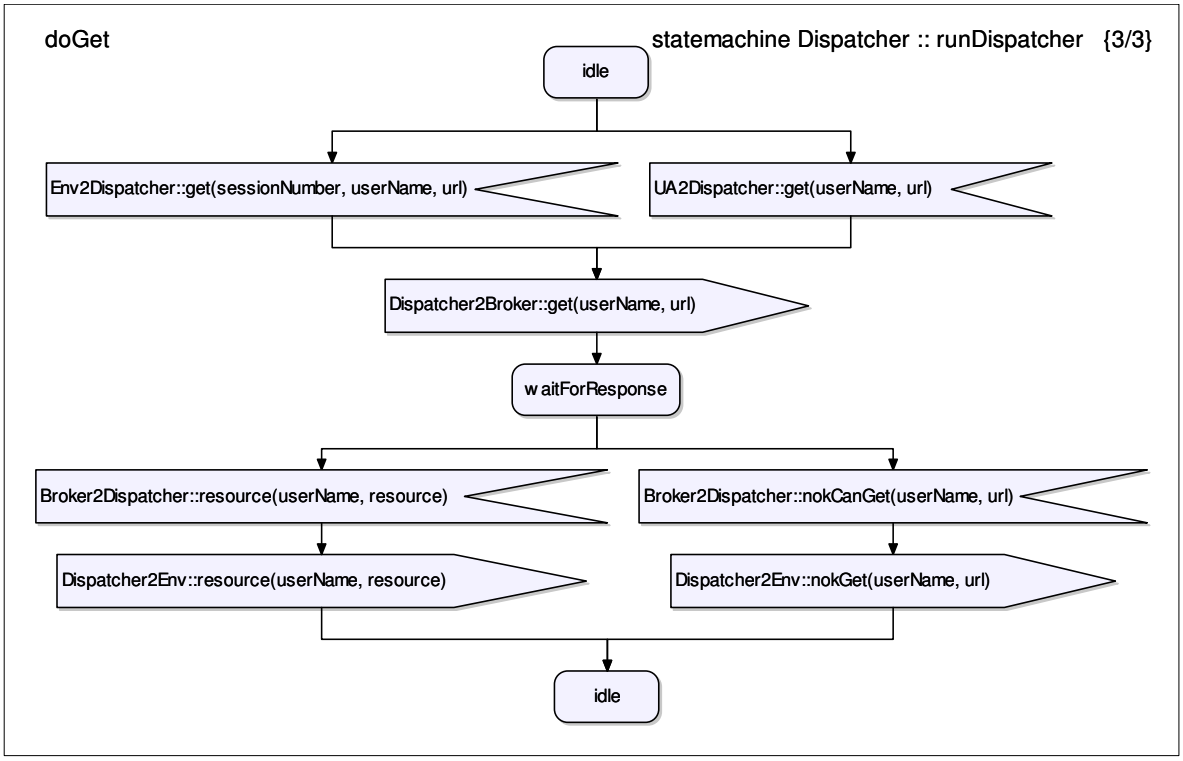


Figure 106. Diagramme SDL de l'action « Browse » pour la classe aiguilleur

Le comportement du courtier est présenté dans le diagramme à la SDL de la *Figure 107*.

Lorsque le courtier reçoit cette requête, il demande au gestionnaire de session si elle peut être satisfaite (message `Broker2SM::canGet(userName,url)`). Dans le cas négatif, il envoie à l'aiguilleur le message `Broker2Dispatcher::nokCanGet(userName,url)`. Dans le cas positif, il demande au gestionnaire de navigation de rapatrier la ressource (message `Broker2BM::get(url)`), et une fois la réponse obtenue, il la relaie à l'aiguilleur (message `Broker2Dispatcher::resource(userName,translatedResource)`) et demande au gestionnaire de session d'informer tous les applets des utilisateurs qui sont actuellement synchronisés avec l'utilisateur qui vient de faire la requête de navigation de charger le même URL dans leur navigateur (message `Broker2SM::synchronize(userName,url)`).

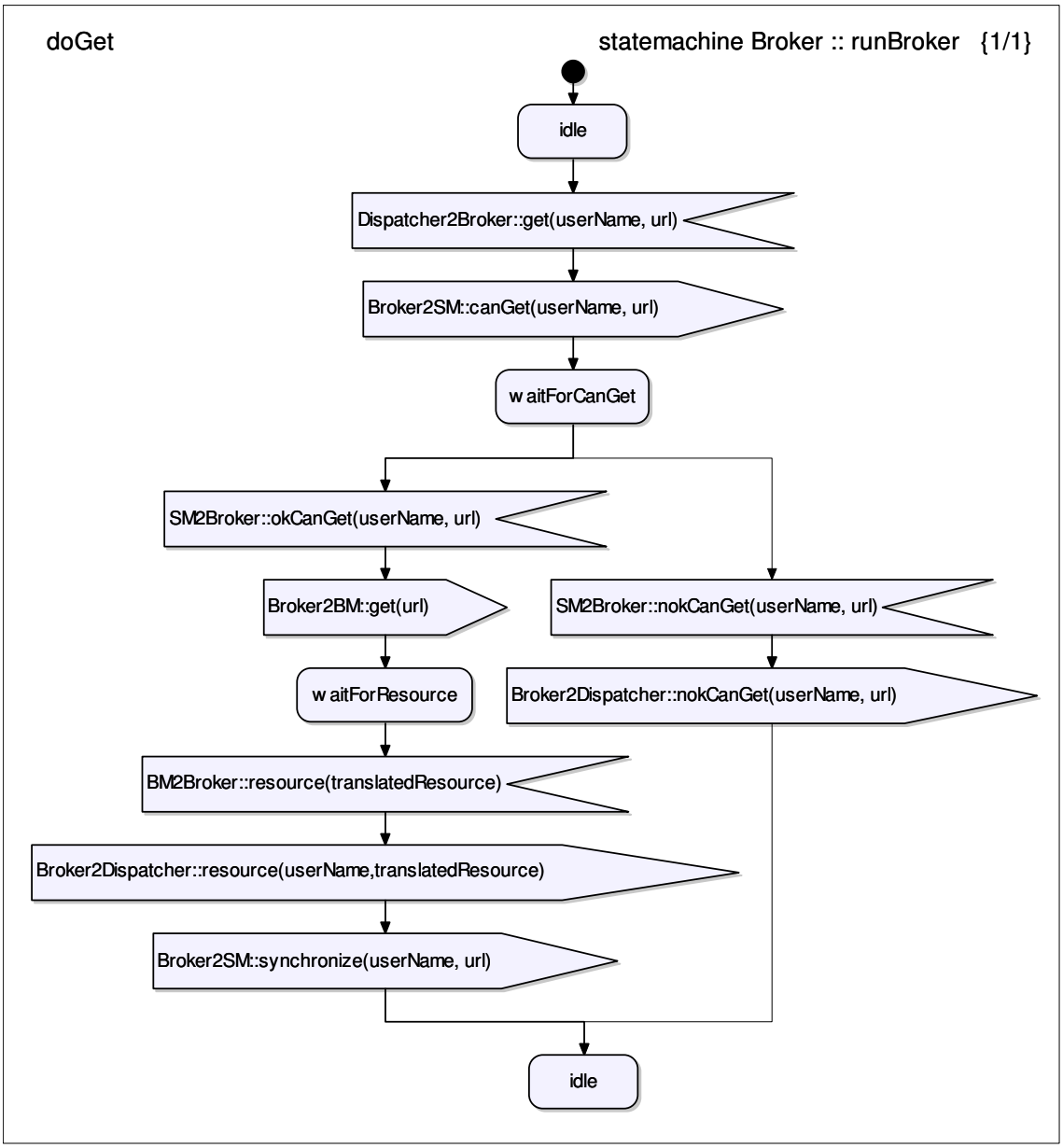


Figure 107. Diagramme SDL de l'action « Browse » pour la classe courtier

Le gestionnaire de navigation, dont le comportement est décrit par le diagramme à la SDL de la *Figure 108*, vérifie auprès du module de cache local, lors de la réception de la requête de navigation, vérifie si la ressource demandée a déjà été rapatriée auparavant, et que la copie locale n'est pas périmée. Dans l'affirmative la copie locale est récupérée et envoyée en réponse à la requête. Dans la négative, il demande au module de rapatriement d'aller chercher la ressource (message `BM2RMod::get(url)`), qui, une fois arrivée, est traduite par le module de traduction puis stockée dans le système de cache local, et est finalement envoyée en réponse à la requête de navigation (message `BM2Broker::resource(translatedResource)`).

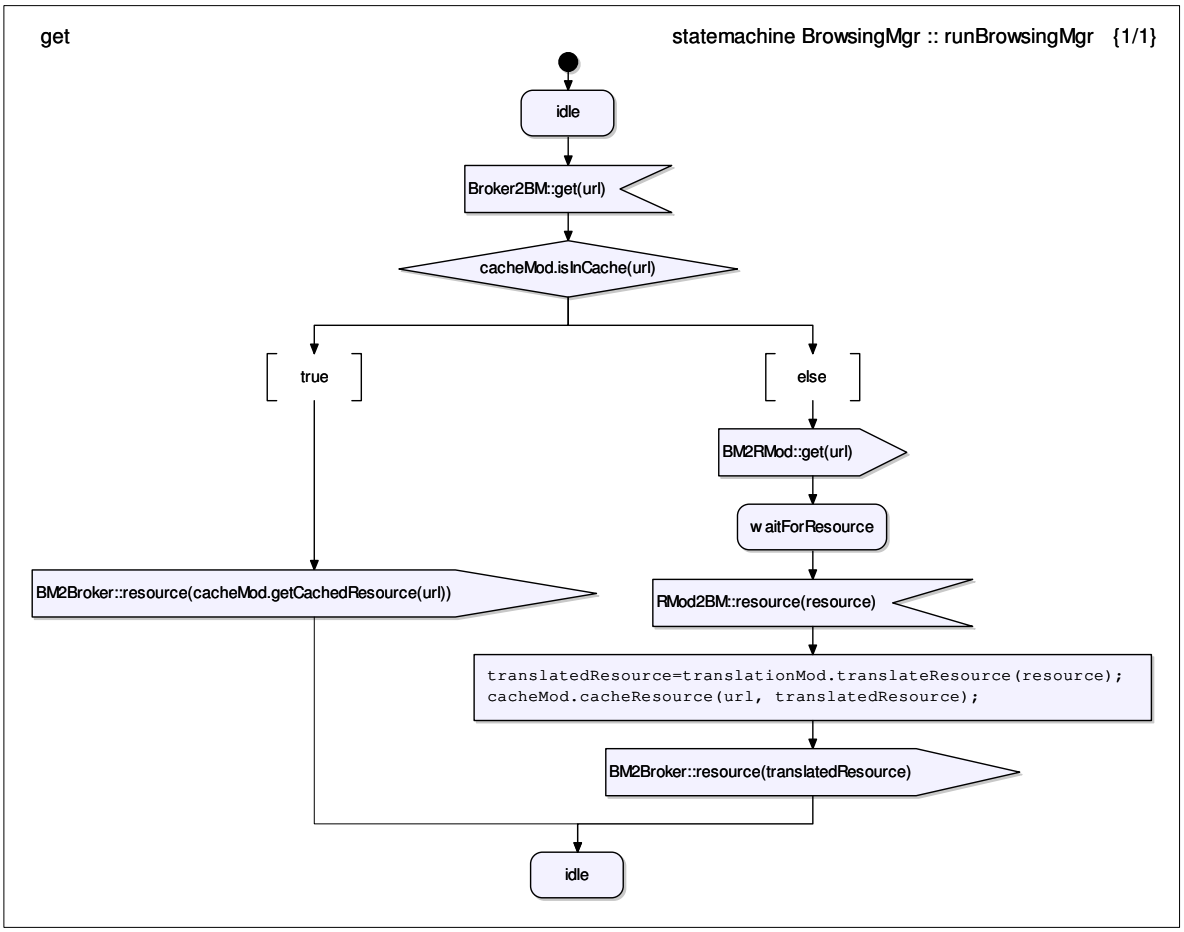


Figure 108. Diagramme SDL de l'action « Browse » pour la classe aiguilleur

IV.5.2. Comparaison des scénarios engendrés par Tau G2 à ceux initialement proposés

A partir de l'ensemble des comportements (ceux que nous venons de décrire les autres), nous avons exécutés diverses simulations en vue de valider l'architecture de notre système. Nous avons ainsi engendré des diagrammes de séquences qui peuvent être comparés aux diagrammes de séquences correspondants aux scénarios initialement proposés dans le paragraphe IV.4.2.

Ainsi, la *Figure 109* et la *Figure 110* présentent les diagrammes de séquences engendrés par simulation qui correspondent à l'entrée d'un utilisateur dans une session et aux différentes situations d'erreur envisagées.

Si nous analysons ces deux diagrammes de séquences et que nous les comparons aux diagrammes de séquence de la *Figure 74*, nous pouvons constater que le comportement réalisé par simulation correspond effectivement au comportement prévu. Dans le but de simplifier la représentation, nous remplaçons des groupes d'actions par des références (« ref »). Ainsi dans la *Figure 110* l'entrée d'un utilisateur de nom d'utilisateur « Memo » est représenté par la référence « `logIn(1, "Memo", "Teacher", "teachpass")` ».

La création de relations de synchronisation au moyen des actions « I_Follow_You » et « You_Follow_Me » est présentée dans la *Figure 111* et la *Figure 112*. Par souci de simplification, les cas d'échec ne sont pas considérés dans ce document.

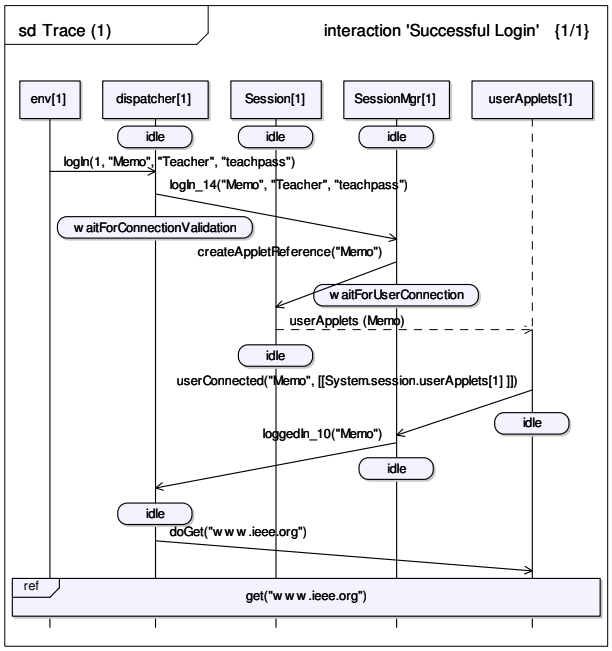


Figure 109. Simulation de l'entrée d'un utilisateur dans une session

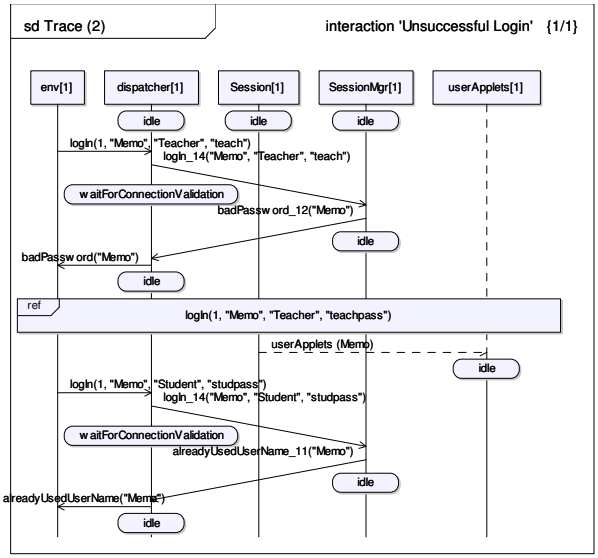


Figure 110. Simulation de tentatives infructueuses d'entrée d'un utilisateur dans une session

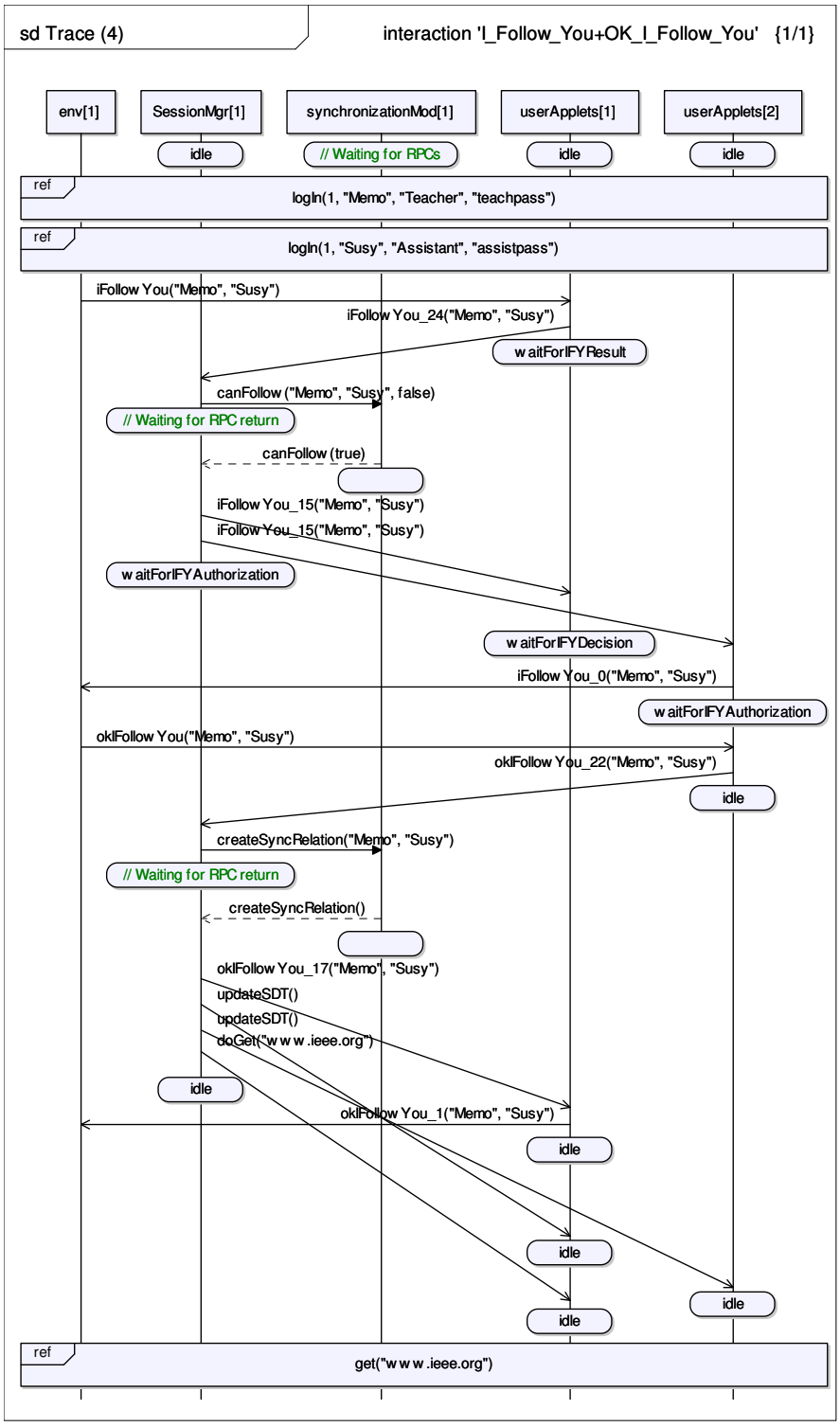


Figure 111. Simulation de la création d'une relation de synchronisation par l'action « I_Follow_You »

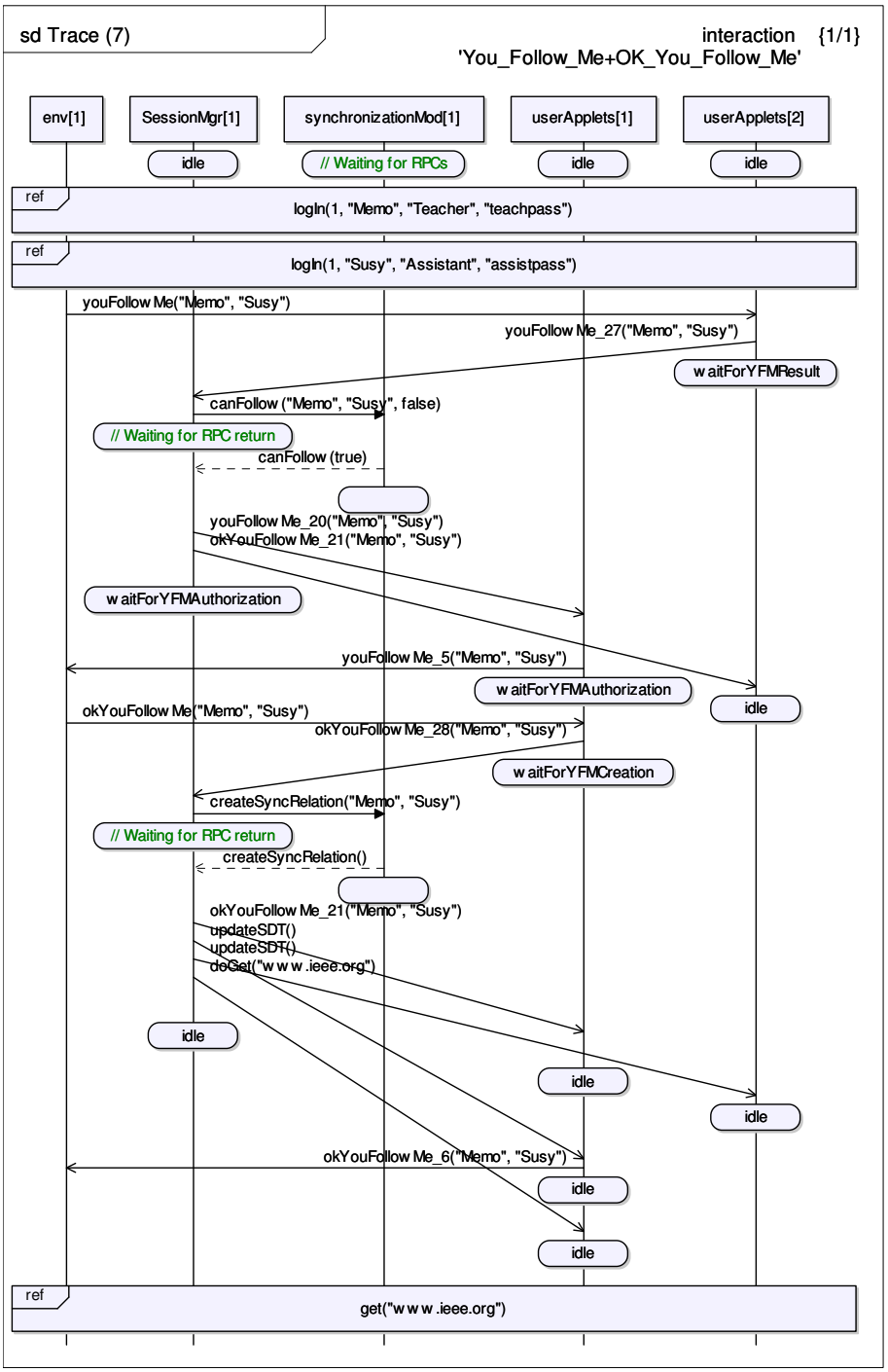


Figure 112. Simulation de la création d'une relation de synchronisation par l'action « You_Follow_Me »

Nous considérons finalement les cas des actions de navigation sur le Web. Dans un premier temps, *Figure 113*, nous présentons la navigation d'un seul utilisateur, qui est asynchrone, en faisant l'hypothèse que la ressource demandée n'a pas encore été rapatriée et n'est donc pas présente dans le cache local.

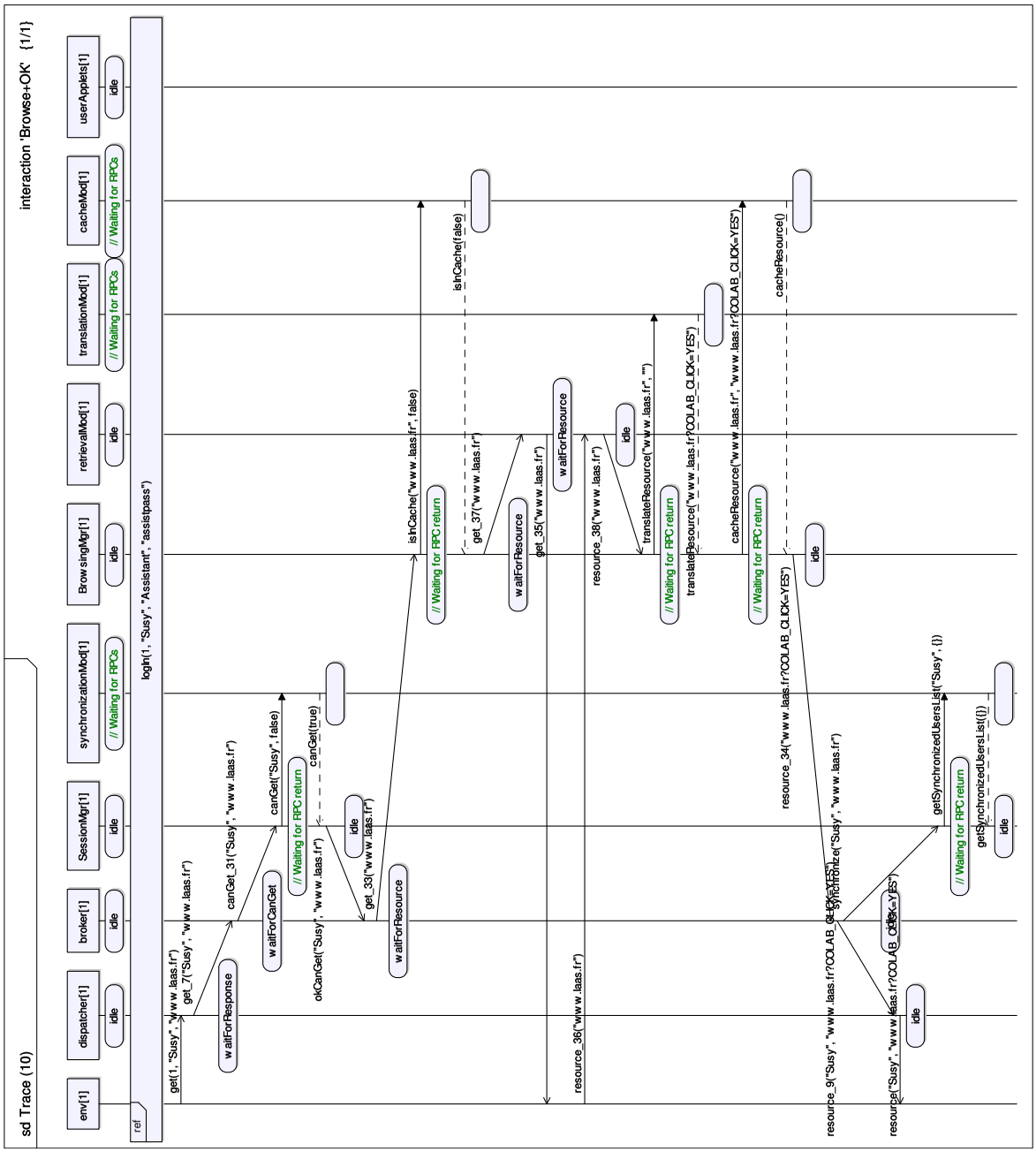


Figure 113. Simulation de la navigation d'un utilisateur asynchrone

Dans le cas de deux utilisateurs synchronisés, le fait que l'utilisateur asynchrone exécute une action de navigation conduit nécessairement à une action de navigation équivalente dans le navigateur de l'utilisateur synchrone, comme cela est illustré dans la *Figure 114*.

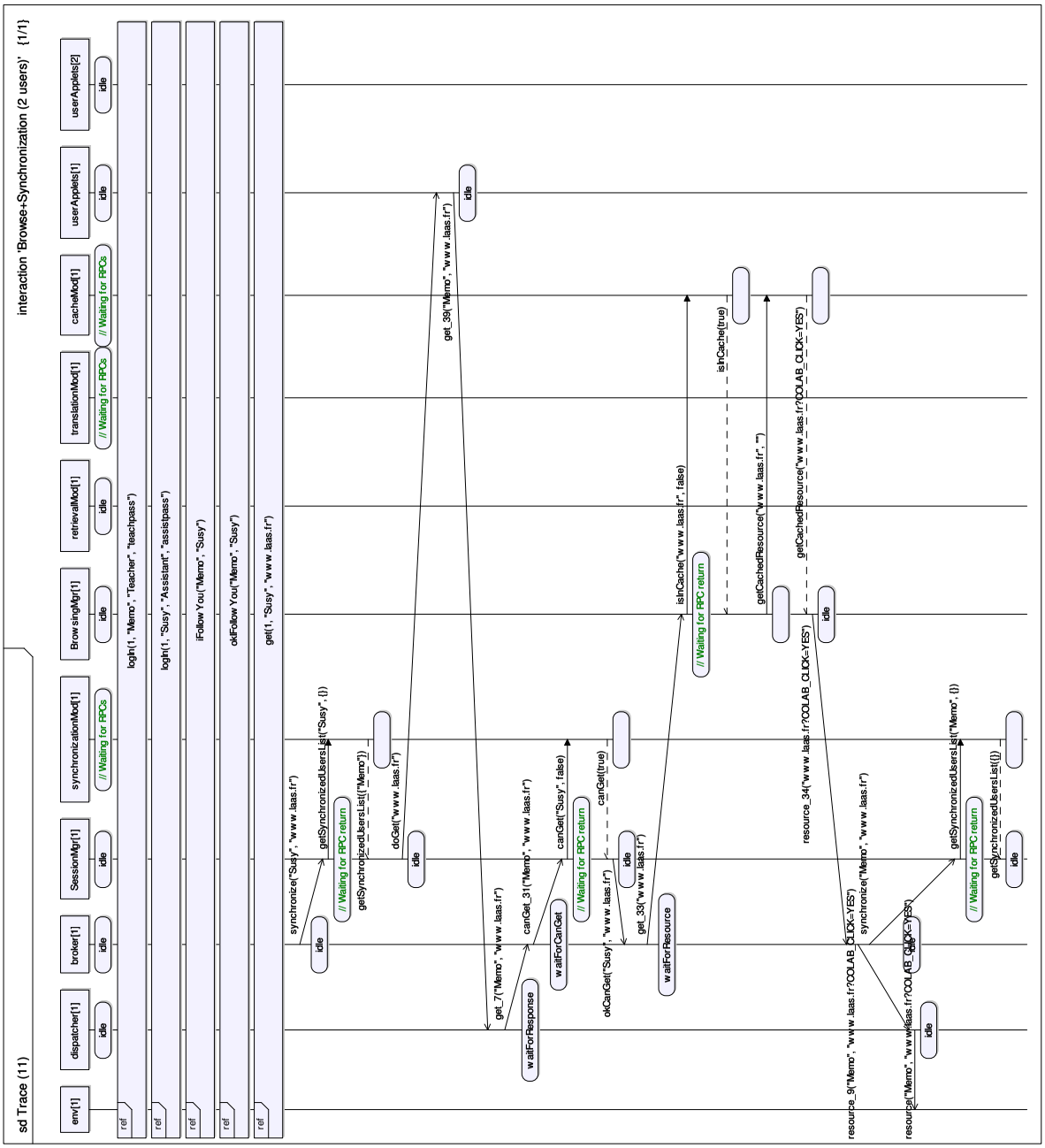


Figure 114. Simulation de la synchronisation de la navigation pour deux utilisateurs

Dans cette dernière figure, nous pouvons voir comment la ressource est récupérée du cache local puisqu'elle a été déjà rapatriée d'un serveur Web distant au moment de la réalisation de la requête de l'utilisateur asynchrone.

IV.6. Conclusion

Dans ce chapitre nous avons décrit en détail l'architecture du système de navigation coopérative que nous proposons en utilisant une méthode de modélisation s'appuyant sur le profil UML/SDL supporté par l'outil Tau G2 de Telelogic. Nous avons validé cette architecture au moyen de plusieurs simulations. Cette validation vient en complément de la vérification formelle du modèle de synchronisation abstrait que nous avons développée dans le chapitre précédent.

La modélisation de l'architecture selon le profil UML/SDL utilisé est le point de départ de l'implémentation et du déploiement de notre système de navigation coopérative CoLab que nous allons décrire en détail dans le chapitre suivant. 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 entamé ce développement, Tau G2 ne disposait pas d'un 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 V

Implémentation et déploiement de CoLab

V.1. Introduction

Nous allons présenter en détail différents aspects techniques concernant l'implémentation de CoLab, l'outil logiciel que nous avons développé dans le but de mettre en œuvre le modèle de navigation coopérative proposé et détaillé dans les chapitres précédents. Dans la version actuelle et opérationnelle de l'outil, nous n'avons pas implémenté toutes les fonctionnalités proposées dans le modèle, mais un sous-ensemble suffisamment représentatif pour valider les idées que nous avons proposées.

Ce chapitre est organisé de la manière suivante : dans un premier paragraphe, après avoir présenté le cadre général de l'implémentation de la plate-forme de navigation coopérative, nous développons en détail les aspects techniques concernant l'implémentation de CoLab; dans un deuxième paragraphe nous présentons des résultats pour évaluer les performances de CoLab avec un serveur Proxy centralisé; dans un troisième paragraphe, nous montrons comment mettre à disposition le service de navigation coopérative sous la forme d'un service Web, avant d'envisager, dans le paragraphe suivant, de distribuer CoLab sur plusieurs serveurs Proxy.

V.2. L'implémentation de CoLab

CoLab est l'outil que nous proposons pour créer un environnement permettant de naviguer de manière coopérative sur la Web. Nous allons maintenant présenter les aspects relatifs à l'implémentation de CoLab, en détaillant les principes de base de son implémentation, de son déploiement et de son utilisation.

Après une analyse de plusieurs plates-formes logicielles, nous avons, pour des raisons de portabilité, fait le choix du langage de programmation Java [Java] et de la technologie Servlets [Servlets] [Hall-01] [Hunter-01] de Sun® Microsystems. Cette technologie offre aux développeurs Web un mécanisme simple pour déployer des applications Web autonomes sans avoir à installer un serveur Web. Un Servlet peut être vu conceptuellement comme une Applet exécutée du côté serveur, mais sans les limitations inhérentes aux programmes CGI (Common Gateway Interface – interface commune d'entrée). Les Servlets ont accès à toute la famille des APIs Java, et peuvent donc être facilement intégrés dans des environnements de développement et d'exécution Java [Java]. Le fait que les Servlets intègrent tout un ensemble de librairies pour traiter des requêtes HTTP, a grandement facilité le développement de CoLab.

Le container de Servlets choisi pour l'implémentation du serveur Proxy de CoLab est celui proposé par le projet Jakarta de Apache : Jakarta Tomcat [Tomcat]. Au moment

de notre choix, la dernière version disponible était la version 3.3 qui implémente la spécification 2.2 des Servlets. La version actuellement disponible de Jakarta Tomcat est la version 5, mais depuis la version 4, le container de Servlets s'appuie sur une toute nouvelle technologie appelée Catalina, qui n'est malheureusement pas compatible avec Tomcat 3.3. Étant donné qu'au moment du déploiement de la version 4, nous avions déjà pas mal avancé l'implémentation de notre prototype, nous avons décidé de ne pas changer de version pour l'instant.

Concernant l'implémentation des mécanismes de communication à l'intérieur de notre plate-forme, nous avons choisi d'utiliser JSDT (Java Shared Data Toolkit – trousse à outils de données partagées de Java) [JSDT-20]. Cet outil offre au développeur l'abstraction de base d'une session (i.e. groupes d'objets associés à des modes de communication), et il est capable de gérer des communications multipoint full-duplex entre un nombre arbitraire d'entités raccordées à un réseau de technologie quelconque. JSDT offre de plus la possibilité de créer des tableaux d'octets (ByteArrays) partagés dont les valeurs peuvent être consultées ou modifiés par les membres d'une session, et que nous avons utilisé pour tenir à tous les clients informés des modifications effectuées à l'état de synchronisation de la session.

JSDT a été développé à l'origine par Sun Microsystems, mais depuis fin 2003 le développement a été arrêté au niveau de la version 2.0. Ce n'est que mi-2004 que le projet a été repris avec une licence LGPL (Lesser General Public License), renommé SDT for Java Technology [JSDT-21]. La version actuelle est la 2.1. Nous continuons pour l'instant à travailler avec la version JSDT 2.0 puisque, à l'heure actuelle, la nouvelle version est encore en cours de développement.

Finalement, en ce qui concerne le côté client de notre application et donc le navigateur Web utilisé par les utilisateurs de CoLab, nous avons fait le choix de la technologie Java au moyen d'Applets Java. L'intégration de JSDT dans des Applets Java est complètement transparente.

CoLab s'appuie, à l'heure actuelle, sur un serveur Proxy dans lequel il est possible de configurer différentes sessions coopératives. A travers ce serveur Proxy nous gérons à la fois la synchronisation des utilisateurs et la synchronisation de la navigation.

La raison pour laquelle nous avons choisi d'implémenter CoLab en nous appuyant sur une approche de serveur Proxy est que, pour arriver à capturer les actions de navigation des utilisateurs, le moyen le plus simple consiste à configurer le navigateur de chaque utilisateur pour que certaines requêtes soient envoyées directement au serveur Proxy. Pour l'instant, toutes les facilités liées à l'implémentation du mécanisme de synchronisation sont également hébergées dans ce serveur Proxy.

Dans le but de mieux illustrer la façon dont les différents composants sont mis en relation, dans la *Figure 115* nous présentons les côtés serveur Proxy et client.

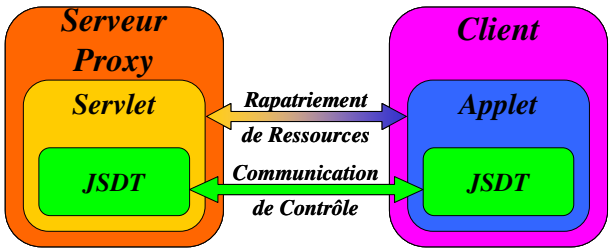


Figure 115. Principaux composants logiciels du serveur Proxy et du client

V.2.1. Le processus de traduction des ressources

Chaque fois qu'un utilisateur clique sur un hyperlien ou saisit un URL dans la barre de navigation de son navigateur, une requête HTTP est créée et envoyée au serveur Web désigné par l'URL. Si la réponse à cette requête est une page Web au format HTML, elle contient probablement des références à d'autres ressources internes à la page (par exemple, des cadres, des images, des sons, etc.). Une fois le chargement de cette page terminé, une requête HTTP est envoyée au serveur pour chacune des ressources internes associées à cette page. Toutes ces requêtes HTTP, celle relative à la page Web initiale et celles relatives aux ressources internes, ont strictement la même structure et il est techniquement impossible de savoir si une requête HTTP est le résultat d'une action de navigation explicite de l'utilisateur ou s'il s'agit du rapatriement d'une ressource interne à une page Web.

Dans la mise en œuvre de notre système de navigation coopérative, il est impératif de pouvoir distinguer ces deux types de requête. Ainsi, lorsqu'un utilisateur *i*, asynchrone, demande le rapatriement d'une page Web, il est nécessaire de forcer la même requête HTTP au niveau des navigateurs de tous les autres utilisateurs synchronisés avec *i*. Par contre, il ne faut pas synchroniser les requêtes HTTP associées au rapatriement des ressources internes à cette page Web.

Plusieurs mécanismes peuvent être envisagés pour distinguer les requêtes HTTP issues d'une action de navigation explicite d'un utilisateur de celles associées au rapatriement de ressources internes associées à une page Web, comme par exemple :

- l'utilisation d'applets signées ;
- l'utilisation de Plug-ins ;
- la traduction des ressources codées en HTML.

Une applet classique s'exécute par défaut dans un environnement sécurisé depuis lequel elle ne peut accéder aux ressources internes de l'ordinateur où elle s'exécute. Une applet signée peut, au contraire, avoir accès à absolument toutes les ressources de la machine où elle s'exécute. En utilisant des applets signées, il est possible de détecter toutes les actions exécutées par l'utilisateur de la machine, donc en particulier ses actions de navigation (par exemple, des clics sur des hyperliens). L'inconvénient majeur est que cette approche fait courir beaucoup de risques de sécurité, sauf si on met en place une politique de sécurité à base de certificats qui est lourde à mettre en œuvre et à déployer.

Les Plug-ins sont des programmes qui sont exécutés dans l'environnement d'exécution d'un navigateur. Nous pourrions déployer des Plug-ins pour détecter toutes les actions des utilisateurs, y compris leurs actions de navigation. Le principal inconvénient de cette approche est qu'il faudrait développer un Plug-in spécifique par type de navigateur considéré ce qui réduirait d'autant la flexibilité de la plate-forme de navigation coopérative.

La technique que nous avons choisie, qui consiste à traduire les ressources HTML, offre le plus de flexibilité sans mettre en péril la sécurité des utilisateurs utilisant la plate-forme de navigation coopérative ni imposer de nouveaux composants dans les navigateurs. Cette technique de traduction consiste à introduire, sous forme de paramètres, des informations de contrôle dans chaque URL définissant un hyperlien. Elle garantit la généralité de notre implémentation, car CoLab peut être utilisé avec n'importe quel navigateur Web, du moment qu'il est conforme aux standards HTTP et HTML du W3C [W3C].

V.2.1.1. Le traitement des hyperliens

La technique de traduction des hyperliens consiste tout d'abord à identifier, à l'intérieur de chaque page Web rapatriée, toute référence à un hyperlien. Pour cela, il suffit

d'identifier une balise HTML <A> et de chercher l'attribut « HREF » qui définit l'URL de destination de l'hyperlien. Plusieurs cas de traduction sont considérés dans les paragraphes suivants.

V.2.1.1.1. Les hyperliens classiques

Le cas de traduction le plus simple concerne les hyperliens classiques. La technique de traduction consiste à ajouter à l'URL de destination de l'hyperlien le paramètre « COLAB_CLICK » avec une valeur « YES », comme cela est illustré dans l'exemple de la *Figure 116*. Ce paramètre permettra de détecter les actions de navigation explicites à l'initiative des utilisateurs.

URL original :

```
<A HREF="http://www.laas.fr/">Le LAAS</A>
```

URL traduit :

```
<A HREF="http://www.laas.fr/?COLAB_CLICK=YES">Le LAAS</A>
```

Figure 116.Traduction des hyperliens : URL sans paramètre

Dans cet exemple, le paramètre « COLAB_CLICK=YES » est ajouté à l'URL original. Suivant les conventions de codification des URLs, la chaîne de caractères contenant les paramètres est séparée du reste de l'URL par l'utilisation d'un caractère '?'.

Il faut bien évidemment tenir compte des hyperliens qui contiennent dans leur définition d'origine des paramètres, de telle façon que la modification introduite n'interfère pas avec le fonctionnement normal de l'hyperlien. Dans un tel cas, le paramètre de contrôle est ajouté à la fin de la chaîne de caractères caractérisant les paramètres de l'URL original. Ceci est illustré dans l'exemple de la *Figure 117*.

URL original :

```
<A href="http://www.leguide.com/fetes/noel.htm?w=1">Noel</A>
```

URL traduit :

```
<A href="http://www.leguide.com/fetes/noel.php?w=1&COLAB_CLICK=YES">Noel</A>
```

Figure 117. Traduction des hyperliens : URL contenant des paramètres

V.2.1.1.2. Les hyperliens vers des points d'ancrage à l'intérieur d'une page Web

Quand l'utilisateur clique sur un hyperlien contenant une référence à un IPL (Intra-Page Link – lien à l'intérieur de la page), la requête est envoyée à un serveur. Cependant l'IPL lui-même n'est jamais envoyé dans la requête. Cette information est traitée exclusivement au niveau du navigateur. Il est donc, dans notre cas, impossible de détecter cette information et de réaliser la synchronisation éventuellement demandée. Pour résoudre ce problème, nous utilisons un autre paramètre, appelé « COLAB_IPL », de valeur égale à la chaîne de caractères qui se trouve après le caractère '#' dans la définition de l'hyperlien. Nous illustrons ce cas de traduction dans la *Figure 118*.

URL original :

```
<A HREF=" http://java.sun.com/j2se/1.5.0/docs/api/java/java/lang/Object.html#
finalize()">finalize</A>
```

URL traduit :

```
<A HREF=" http://java.sun.com/j2se/1.5.0/docs/api/java/java/lang/Object.html#
finalize()?COLAB_CLICK=YES&COLAB_IPL=finalize()">finalize</A>
```

Figure 118. Traduction des hyperliens : traitement des IPL

Dans cet exemple, nous pouvons observer que l'information concernant l'IPL est dupliquée, l'une est destinée au navigateur où l'action de navigation a été réalisée, l'autre au serveur Proxy pour qu'il puisse la diffuser aux navigateurs des utilisateurs devant synchroniser leur navigation.

Dans le cas de la définition d'un IPL pointant au sein de la même page, aucune requête n'est envoyée au serveur puisque la ressource est déjà chargée et affichée dans le navigateur du client. L'activité de navigation n'est donc pas détectable au niveau du serveur Proxy et aucune synchronisation ne peut être réalisée. Pour forcer l'envoi de la requête au serveur, nous remplaçons la référence relative de l'URL par la référence absolue correspondante, ce qui est illustré dans l'exemple de la *Figure 119*.

URL original :

```
<A HREF="#field_summary">FIELD</A>
```

URL traduit :

```
<A HREF="http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Byte.html#
field_summary?COLAB_CLICK=YES&COLAB_IPL=field_summary">FIELD</A>
```

Figure 119. Traduction des hyperliens : cas d'un URL relatif

V.2.1.1.3. Autres cas de traduction

Il existe d'autres cas où la traduction est nécessaire. C'est notamment le cas des éléments `<BASE>` et `<META>`. Dans le premier cas, cet élément sert à définir l'URL de base par défaut pour toute référence relative contenue dans une page Web, de telle façon qu'à chaque fois que le navigateur trouve un URL relatif, il le modifie systématiquement en ajoutant à la référence relative l'URL de base. Dans le second cas, cet élément sert à envoyer au navigateur du client une information qui doit être traitée de manière similaire aux en-têtes. En ce qui nous concerne, il existe un modificateur utilisé pour forcer le chargement d'un URL spécifique, qui est présenté dans l'exemple de la *Figure 120*.

Code original :

```
<META HTTP-EQUIV=Refresh CONTENT="0;http://oglobo.globo.com/online/default.asp">
```

Code traduit :

```
<META HTTP-EQUIV=Refresh
CONTENT="0;http://oglobo.globo.com/online/default.asp?COLAB_CLICK=YES">
```

Figure 120. Traduction de la balise <META>

Ici lorsque le navigateur trouve cette définition, il charge immédiatement l'URL défini dans « CONTENT », d'où le besoin de traduire cet URL.

V.2.1.2. Le traitement des formulaires

La technique de traduction dans le cas des formulaires est un peu plus complexe que dans le cas des hyperliens. Pour l'expliquer considérons l'exemple de la *Figure 121*.

Code original :

```
<FORM action="http://www.laas.fr/prog/adduser" method="get">
  Prénom: <INPUT type="text" name="prenom"><BR>
  Nom: <INPUT type="text" name="nom"><BR>
  Courriel: <INPUT type="text" name="courriel"><BR>
  <INPUT type="radio" name="sex" value="Homme"> Homme<BR>
  <INPUT type="radio" name="sex" value="Femme"> Femme<BR>
  <INPUT type="submit" value="Envoyer"> <INPUT type="reset" value="Réinitialiser">
</FORM>
```

Code traduit :

```
<FORM action="http://COLAB_PRPX_www.laas.fr/prog/adduser" method="get">
  <INPUT name="COLAB_CLICK" value="YES" type="hidden">
  Prénom: <INPUT type="text" name="prenom"><BR>
  Nom: <INPUT type="text" name="nom"><BR>
  Courriel: <INPUT type="text" name="courriel"><BR>
  <INPUT type="radio" name="sex" value="Homme"> Homme<BR>
  <INPUT type="radio" name="sex" value="Femme"> Femme<BR>
  <INPUT type="submit" value="Envoyer"> <INPUT type="reset" value="Réinitialiser">
</FORM>
```

Figure 121. Traduction de formulaires

Dans le cas des formulaires, il n'est pas possible de définir des paramètres directement dans l'URL comme cela était possible avec les hyperliens ; or, comme nous l'avons déjà expliqué, il nous faut introduire le paramètre « COLAB_CLICK=YES » pour pouvoir détecter l'action de navigation.

Pour résoudre ce problème nous utilisons une astuce qui consiste à introduire une entrée cachée (élément HTML <INPUT>), qui contient la définition du paramètre qui nous est nécessaire pour reconnaître une action de navigation d'un utilisateur.

Cette modification est adaptée lorsque la méthode utilisée dans le formulaire est la méthode « GET ». Dans le cas de la méthode « POST » les paramètres ne sont pas envoyés dans la requête elle-même, mais dans un paquet à part. Si c'est le cas, le paramètre « COLAB_CLICK » ne peut alors pas être détecté au niveau du serveur Proxy, empêchant ainsi toute synchronisation. Pour résoudre ce problème, nous avons choisi de modifier le nom du serveur au niveau de l'URL en lui ajoutant le préfix « COLAB_PRPX_ ». Ainsi nous garantissons que, indépendamment de la méthode utilisée « GET » ou « POST », l'action de navigation sera correctement détectée par notre système.

V.2.1.3. Le traitement des cadres

La traduction des pages Web contenant des cadres se fait en deux étapes. La première étape concerne la traduction de la page principale, c'est à dire celle qui contient la définition de l'ensemble de cadres (« FRAMESET »). La deuxième étape concerne la traduction de chacune des pages contenues dans chacun des cadres définis.

La traduction de la page principale consiste à ajouter à l'URL associé à chaque cadre le paramètre « COLAB_TARGET », avec une valeur égale au nom du cadre où la ressource est chargée. Cette traduction est nécessaire afin de, au moment du rapatriement de chacune des ressources associées à chacun des cadres, pouvoir récupérer le nom de cadre dans lequel la ressource doit être affichée et introduire cette information dans la traduction des hyperliens de cette ressource. Dans l'exemple de la *Figure 122* nous illustrons cette technique de traduction.

Code original :

```
<FRAMESET cols="20%,80%" title="">
  <FRAMESET rows="30%,70%" title="">
    <FRAME src="overview-frame.html" name="packageListFrame" title="All Packages">
    <FRAME src="allclasses-frame.html" name="packageFrame" title="All classes and
interfaces">
  </FRAMESET>
  <FRAME src="overview-summary.html" name="classFrame" title="Package, class and interface
descriptions" scrolling="yes">
</FRAMESET>
```

Code traduit :

```
<FRAMESET cols="20%,80%" title="">
  <FRAMESET rows="30%,70%" title="">
    <FRAME name="packageListFrame" src="overview-frame.html?COLAB_TARGET=packageListFrame"
title="All Packages">
    <FRAME name="packageFrame" src="allclasses-frame.html?COLAB_TARGET=packageFrame"
title="All classes and interfaces">
  </FRAMESET>
  <FRAME name="classFrame" src="overview-summary.html?COLAB_TARGET=classFrame"
title="Package, class and interface descriptions" scrolling="yes">
</FRAMESET>
```

Figure 122. Traduction de la définition d'un ensemble de cadres (FRAMESET)

Une fois que la définition des cadres a été chargée, le navigateur rapatrie les pages définies pour chacun des cadres définis. Nous appliquons ici la technique de traduction classique des hyperliens (ajout du paramètre « COLAB_CLICK=YES ») et ajoutons également à chaque hyperlien le paramètre « COLAB_TARGET » avec comme valeur le nom du cadre où est affichée la ressource. Dans le cas où un ou plusieurs cadres n'ont pas de nom, le système définit un nom par défaut qui est le nom de la ressource chargée dans le cadre.

Au niveau de la définition des hyperliens, il est possible de définir que la ressource associée doit être présentée dans un cadre spécifique, ce qui se fait par utilisation de l'attribut « TARGET » dans la définition de l'hyperlien. Si cet attribut a une des valeurs _blank », « _parent » ou « _top », nous remplaçons le nom du cadre de destination par « COLAB_NAVIGATION », qui est le nom par défaut du cadre de navigation de CoLab, ceci afin de ne pas modifier la structure de cadres propre à CoLab. Si cet attribut a pour valeur un nom de cadre, nous ne le modifions pas.

V.2.1.4. Le traitement des en-têtes HTTP

En ce qui concerne la version de HTTP utilisée, indépendamment de la version utilisée par le navigateur, nous forçons toujours, tant du côté des réponses envoyées aux clients que des requêtes envoyées au serveur Proxy, l'utilisation de HTTP 1.0, ce qui est conforme aux recommandations du W3C en présence d'un serveur Proxy.

Dans un environnement de navigation coopérative, où l'une des caractéristiques est l'hétérogénéité des navigateurs utilisés et où l'on souhaite contrôler la manière dont les ressources rapatriées sont stockées dans le cache local du navigateur, il est nécessaire de bien gérer les en-têtes pour qu'ils n'interfèrent pas avec le fonctionnement de la plate-forme. Nous avons identifié des en-têtes de requête et des en-têtes de réponse qui risquent d'interférer avec CoLab.

V.2.1.4.1. Les en-têtes de requête

Certains en-têtes de requête peuvent créer des comportements indésirables au niveau de la plate-forme. Il ne faut donc pas que le serveur Proxy les relaye aux serveurs Web. Ce sont les en-têtes suivants :

- **`User-Agent :`** cet en-tête ne doit pas être pas relayé au serveur pour ne pas l'informer du type de navigateur utilisé, et pour que, en conséquence, la réponse soit la plus générale possible (pas de version spécifique pour un type de navigateur) ;
- **`Accept-Encoding :`** le serveur ne doit pas être informé qu'un navigateur est capable de traiter des ressources transmises en format compressé (par exemple, en format gzip), puisque si la réponse reçue n'est pas en format texte le serveur Proxy ne pourra pas la traduire ;
- **`ETag, If-Match, If-Modified-Since, If-None-Match, If-Unmodified-Since, If-Range :`** puisque l'on désire que chacune des requêtes soit détectée par le serveur Proxy, et que à chaque réponse envoyée aux clients ce soit exactement la même information qui s'affiche, nous empêchons toute mise en cache local par un navigateur. Ces en-têtes ne doivent donc pas être envoyés au serveur pour empêcher que le système de cache local du navigateur n'interfère avec le rapatriement des ressources ;
- **`Connection, Keep-Alive, Proxy-Connection :`** lorsqu'on utilise un serveur Proxy pour le rapatriement de ressources, il est fortement conseillé d'utiliser la version 1.0 de HTTP, donc des connexions non persistantes. Pour cette raison, le serveur Proxy ne doit pas relayer les en-têtes définissant des connexions persistantes.

Tous les autres en-têtes de requête sont relayés sans modification au serveur, puisqu'ils n'interfèrent pas avec le système de navigation coopérative.

V.2.1.4.2. Les en-têtes de réponse

En ce qui concerne les en-têtes de réponse, il est uniquement nécessaire de traiter les en-têtes relatifs à la gestion du cache local du navigateur et ceux relatifs à la gestion de la connexion. Ces en-têtes sont donc également bloqués au niveau du serveur Proxy. Une fois bloqués, le serveur Proxy enverra au client les en-têtes définis dans la *Figure 123*, qui informent le navigateur qu'il ne doit rien stocker dans son cache local et qu'il ne doit pas essayer d'utiliser de connexions persistantes.

```
Connection: close
Cache-Control: no-cache
Pragma: no-cache
```

Figure 123. En-têtes de gestion de cache envoyés systématiquement aux clients

Trois autres en-têtes de réponse doivent également être considérés, à savoir « Content-Type », « Content-Lenght » et « Location ».

L'en-tête « Content-Type » est analysé par le serveur Proxy pour décider du type de ressource qu'il est en train de recevoir comme réponse à sa requête. Quand le serveur Proxy reçoit une ressource dont le type MIME est « text/html », le serveur Proxy lui applique la technique de traduction, sinon la ressource est retransmise au client sans modification.

L'en-tête « Content-Length » informe le navigateur de la taille de la ressource envoyée dans la réponse. Cependant, cet en-tête n'est pas obligatoire, et en cas d'absence, le client continue de recevoir des données jusqu'à la fermeture de la connexion par le serveur. En conséquence, rien ne nous empêche, en cas de rapatriement d'une ressource de type MIME « text/html », de ne pas relayer cet en-tête au navigateur. Ceci est important, puisque lors de l'application de la technique de traduction, la taille de la ressource va bien évidemment être modifiée.

Finalement l'en-tête « Location » indique au navigateur l'URL où il doit aller chercher la ressource dans le cas où elle ne se trouve plus à l'URL défini dans la requête originale (codes de réponse 301 et 302). Une fois cet en-tête reçu par le serveur Proxy, il traduit l'URL selon la technique expliquée précédemment avant de le relayer au navigateur.

V.2.2. Le protocole de synchronisation des utilisateurs

La synchronisation des utilisateurs au sein d'une session de navigation coopérative constitue le noyau de notre proposition. Nous avons validé et implémenté les trois actions de synchronisation de base, à savoir : « I_Follow_You », « You_Follow_Me » et « I_Leave_You ». Remarquons que nous ne faisons pas exactement les mêmes hypothèses concernant la concurrence des actions de synchronisation au niveau de l'implémentation et au niveau de la modélisation (Chapitre III).

Dans la modélisation nous avons empêché, au moyen d'une place d'exclusion mutuelle appelée « Mutex », l'occurrence de toute nouvelle action de synchronisation tant que l'action de synchronisation en cours n'était pas complètement traitée. Ceci a grandement facilité notre modélisation et la vérification formelle associée.

Dans l'implémentation, nous avons mis en œuvre un mécanisme plus lâche, sachant que plusieurs actions de synchronisation peuvent en fait s'exécuter de manière indépendante. Ainsi, nous ne bloquons que les actions de synchronisation à l'initiative des utilisateurs directement concernés par l'action de synchronisation en cours, ce que nous allons développer dans le paragraphe suivant.

V.2.2.1. Gestion de la concurrence des actions de synchronisation

L'état de synchronisation d'une session est géré de manière centralisée par le serveur Proxy, donc toute requête de synchronisation émise par un utilisateur lui est directement envoyée pour traitement.

Au niveau de la représentation de l'état de synchronisation d'une session, nous utilisons deux structures de données en parallèle :

- l'état courant de synchronisation de l'ensemble des utilisateurs de la session, appelé CSDT (Committed SDT), et qui correspond à l'ensemble des SDTs dans la session courante ;
- une structure auxiliaire de données, appelée RSDT (Requested SDTs), servant à stocker temporairement les intentions de synchronisation associées aux actions de synchronisation nécessitant un mécanisme d'autorisation préalable. Les requêtes de synchronisation sont ainsi enregistrées de manière temporaire dans le RSDT, le temps que l'autorisation, l'annulation ou le refus soit exprimé. Dès réception de l'autorisation, le résultat de l'action de synchronisation est enregistré dans le SDT correspondant et donc dans le CSDT. En cas de refus ou d'annulation de la requête, le RSDT est réinitialisé à l'état qu'il avait avant traitement de la requête de synchronisation.

De cette manière lorsque le serveur Proxy reçoit un message l'informant de l'intention d'un utilisateur de créer une relation de synchronisation avec un autre, il vérifie auprès du RSDT si cette intention n'entre pas en conflit avec une autre intention manifestée auparavant. S'il n'y a pas de conflit, la relation de synchronisation est créée dans le RSDT le temps qu'une réponse à cette requête (acceptation, refus ou annulation) arrive. La requête est donc relayée à l'utilisateur concerné pour qu'il puisse s'exprimer.

Si la requête de création de la relation de synchronisation est acceptée, alors la modification précédemment appliquée au RSDT est appliquée également au CSDT. Cette dernière structure est alors diffusée à tous les utilisateurs connectés à la session pour actualiser leurs navigateurs.

Si la requête de création de la relation de synchronisation est refusée ou annulée, le RSDT est remis à l'état où il se trouvait avant la requête de synchronisation, et aucune actualisation n'est réalisée dans les navigateurs des utilisateurs.

Le mécanisme proposé met ainsi en œuvre un mécanisme classique d'engagement à deux phases. À fin d'illustrer ces idées, considérons les CSDT et RSDT de la *Figure 124*.

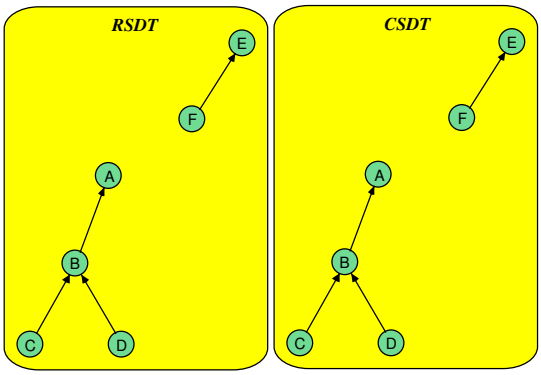


Figure 124. États stables du RSDT et du CSDT

Supposons que l'utilisateur *A* décide de se synchroniser avec l'utilisateur *F*. Au moment de la réception de cette requête, le serveur Proxy modifie le RSDT, comme cela est illustré dans la *Figure 125*.

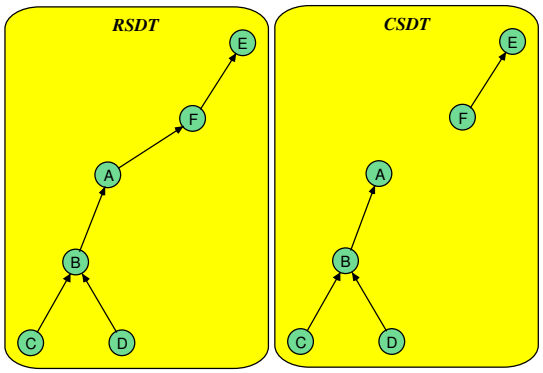


Figure 125. État du RSDT et du CSDT en présence d'une requête de création d'une relation de synchronisation

Notons qu'à ce stade aucun des utilisateurs appartenant à la session, à l'exception des utilisateurs concernés par l'action de synchronisation (c'est-à-dire *A* et *F*), n'est au courant de cet événement, puisque le CSDT n'a pas encore été modifié. Donc, l'utilisateur *E* pourrait, par exemple, essayer de se synchroniser à un autre utilisateur, ce qui pourrait créer des situations d'incohérence à cause de la formation de boucles.

Imaginons donc, avant qu'une autorisation, refus ou annulation de la requête de synchronisation précédente ne soit exprimée, que l'utilisateur *E* décide de se synchroniser avec l'utilisateur *A*. Le serveur Proxy peut facilement se rendre compte que cette deuxième requête conduit à la création d'un état incohérent, et en conséquence ne l'accepte pas. Une telle situation est illustrée dans la *Figure 126*.

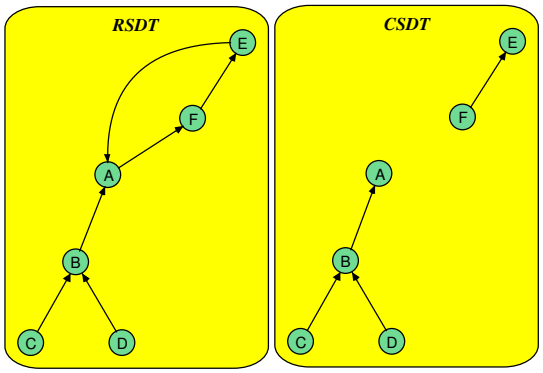


Figure 126. État incohérent du RSDT

V.2.3. Fonctionnement du serveur Proxy

Avant de pouvoir utiliser CoLab, il faut exécuter certaines tâches de configuration, pour pouvoir démarrer ensuite le serveur Proxy. Deux éléments doivent en particulier être configurés : l'URL qui permet d'accéder au fichier de configuration automatique du Proxy, et le fichier de configuration de session, pour chacune des session qui seront mises à disposition.

V.2.3.1. Le fichier de configuration automatique du Proxy

Le fichier de configuration automatique du Proxy, connu également comme fichier PAC (Proxy Automatic Configuration – configuration automatique de Proxy), est un moyen destiné à configurer la plupart des navigateurs Web pour qu'ils traitent les requêtes HTTP de manière sélective en fonction de certaines conditions. Ce fichier est programmé

en utilisant le langage de programmation JavaScript [Jaworsky-99]. Ce fichier contient une seule définition de fonction : « findProxyForURL » avec deux paramètre formels : 'url', qui représente l'URL de la requête sous forme de chaîne de caractères, et 'host', qui représente le nom de l'hôte spécifié dans la requête. Cette fonction qui dépend d'un ensemble de conditions retourne une chaîne de caractères contenant soit l'URL du serveur Proxy à contacter, soit la chaîne de caractères « DIRECT », qui indique au navigateur qu'il ne doit passer par aucun Proxy pour rapatrier une ressource, mais qu'il doit au contraire accéder directement au serveur d'origine. Dans le cas de CoLab, nous utilisons actuellement le fichier PAC décrit dans la *Figure 127*.

```
function findProxyForURL(url, host) {
        if(host.toLowerCase() == "colab.laas.fr")
                return "PROXY banderilla.laas.fr:8090";
        else if(url.toUpperCase().indexOf("COLAB_") >= 0)
                return "PROXY banderilla.laas.fr:8090";
        else
                return "DIRECT";
}
```

Figure 127. Le fichier de configuration automatique de Proxy (fichier PAC)

Nous pouvons identifier ici deux conditions et un cas par défaut.

La première condition a pour but de détecter une requête destinée à un serveur fictif dont le nom d'hôte est « colab.laas.fr ». Ce nom de serveur n'existe pas réellement et il n'est utilisé que dans la procédure d'initialisation de CoLab, comme nous le détaillerons ultérieurement.

La deuxième condition teste l'existence de la chaîne de caractères « COLAB_ ». Cette chaîne de caractères, comme nous l'avons déjà expliqué, est ajoutée par le processus de traduction à tout élément HTML spécifiant un hyperlien ; nous l'utilisons ainsi comme un moyen pour identifier les requêtes devant être obligatoirement traités par le Proxy. Ainsi, lorsque le navigateur trouve une requête contenant cette chaîne de caractères, il relaie la requête vers le serveur Proxy. Dans la *Figure 128* nous illustrons la manière dont ce mécanisme fonctionne.

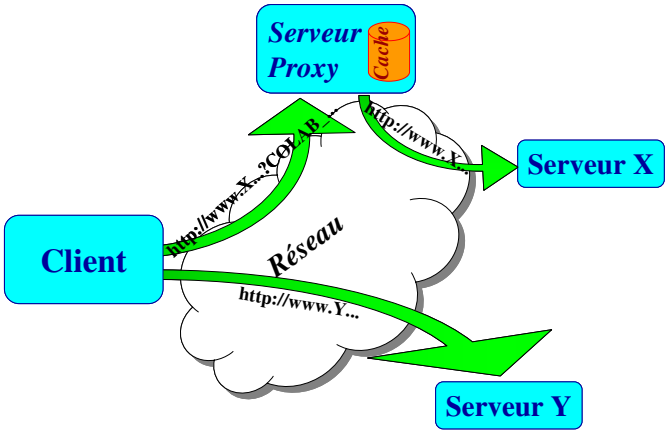


Figure 128. Rôle de la chaîne de caractères « COLAB_ » pour l'accès au serveur Proxy

Dans les deux cas précédents, lorsque l'une des conditions est satisfaite, la requête est relayée vers le serveur Proxy de nom « banderilla.laas.fr », à travers le port TCP 8090. Ce nom d'hôte désigne l'ordinateur où le serveur Proxy de CoLab s'exécute.

Finalement, l'option par défaut indique au navigateur que toute requête, ne satisfaisant aucune des deux conditions précédentes, devra accéder directement au serveur d'origine spécifié dans l'URL de l'hyperlien activé, ce qui implique que l'on n'utilisera donc pas de serveur Proxy.

Le fichier PAC peut être disponible dans un serveur quelconque. S'il se trouve dans le même ordinateur où est installé le serveur Proxy de CoLab, ce fichier doit être disponible en dehors de l'environnement CoLab, c'est-à-dire accessible via un serveur Web traditionnel.

V.2.3.2. Le fichier de configuration de session

Le serveur Proxy de CoLab peut gérer plusieurs sessions en parallèle. Chacune de ces sessions a, par définition, ses propres caractéristiques et son propre espace de travail. Ceci signifie que les sessions sont complètement indépendantes et que, par conséquent, les requêtes de synchronisation et de navigation exécutées au sein d'une session n'ont aucun effet sur les autres sessions.

Les sessions CoLab sont configurées au moyen d'un fichier de configuration codé en XML, grâce auquel les principaux paramètres de la session sont définis. Un exemple de ce type de fichier est présenté dans la *Figure 129*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<colab_session url="http://www.yahoo.com">
   <role_definition>
      <role role_name="Teacher" role_password="teachpass"/>
      <role role_name="Assistant" role_password="assistpass"/>
      <role role_name="Student" role_password="studpass"/>
   </role_definition>
   <role_privileges>
      <canSpy from="Teacher" to="Student"/>
      <canForce from="Teacher" to="Student"/>
      <canSpy from="Assistant" to="Student"/>
   </role_privileges>
</colab_session>
```

Figure 129. Un exemple de fichier de configuration de session

Le nom de session est défini par le nom du fichier contenant la définition de session. À l'intérieur de ce fichier existe une balise de niveau principal : « colab_session », dont le seul paramètre sert à définir l'URL par défaut pour cette session. Cet URL est affiché lorsqu'un utilisateur entre dans la session ou lorsqu'un utilisateur sélectionne le bouton « HOME » dans le GUI de CoLab.

À l'intérieur de la section principale, nous trouvons deux sections dont seule la première est obligatoire. Cette section sert à définir les rôles disponibles pour la session, et les mots de passe associés à chacun des rôles. La section suivante sert à définir d'éventuels privilèges que chaque rôle peut avoir sur d'autres rôles. Rappelons que les deux privilèges définis dans le modèle de synchronisation étendu sont « canSpy » et « canForce », mais ils ne sont pas encore implémentés à l'heure actuelle.

V.2.4. L'accès à une session de navigation coopérative

Une fois le Proxy CoLab démarré, les utilisateurs peuvent accéder à des sessions de navigation coopérative gérées au niveau de ce serveur, après avoir configuré leur navigateur.

V.2.4.1. La Configuration du navigateur

La configuration du navigateur est très simple, et ne consiste qu'à définir l'URL à partir duquel il est possible de rapatrier le fichier PAC contenant la définition d'accès au serveur Proxy de CoLab. Par exemple, dans le cas de Netscape Navigator® [Netscape] il faut accéder, dans le menu principal, à l'option « Edit – Preferences », et dans la fenêtre qui apparaît, dans le menu à gauche, à l'option « Advanced », puis « Proxies ». Là il faut cocher l'option « Automatic proxy configuration URL : », et dans le champ textuel correspondant saisir l'URL du fichier PAC. Dans la *Figure 130* nous présentons cette fenêtre.

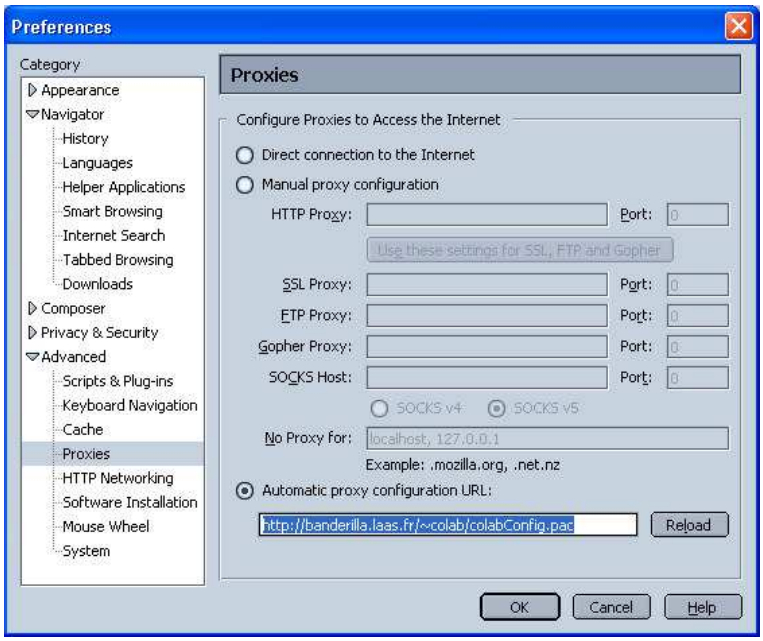


Figure 130. Configuration du fichier PAC avec Netscape Navigator®

Dans le cas de Microsoft Internet Explorer® [MSIE], il faut accéder à l'option « Outils – Options Internet… » du menu principal, puis à l'option « Paramètres réseau… », puis cocher l'option « Utiliser un script de configuration automatique », et saisir l'URL du fichier PAC. Ceci est illustré dans la *Figure 131*.

Finalement, pour le navigateur Opera® [Opera] il faut accéder à l'option « Outils – Préférences » du menu principal, puis choisir la rubrique « Réseau » et cliquer sur le bouton « Serveurs Proxy ». Dans la fenêtre qui s'ouvre alors, il faut cocher l'option « Utiliser la configuration automatique du proxy » et saisir l'URL du fichier PAC. Ceci est illustré dans la *Figure 132*.

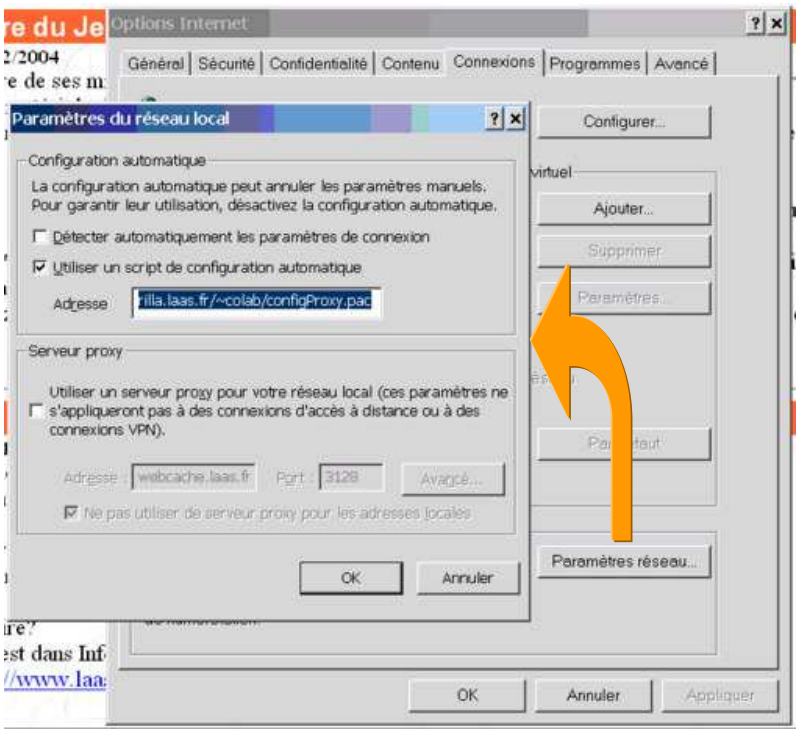


Figure 131. Configuration du fichier PAC avec Microsoft Internet Explorer®

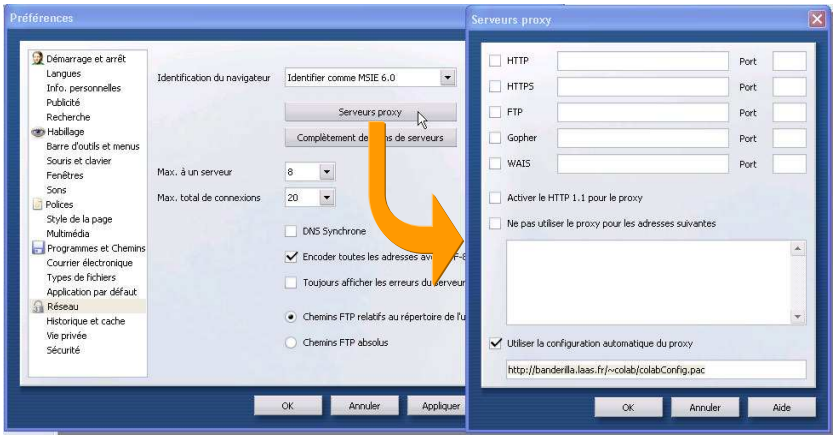


Figure 132. Configuration du fichier PAC dans Opera®

Pour d'autres navigateurs, tel que Firefox® [Firefox], Mozilla® [Mozilla], K-Meleon® [K-Meleon], etc., avec lesquels nous avons testé notre plate-forme de navigation coopérative, la procédure de configuration est très proche de celle utilisée pour Netscape Navigator®.

Une fois le navigateur configuré, il suffit d'accéder à l'URL qui sert de point d'accès à CoLab (i.e. URL spécifié dans la première condition du fichier PAC. Pour le fichier de configuration de la *Figure 127*, l'URL est « http://colab.laas.fr ».

Un autre point important est relatif aux systèmes de blocage des fenêtres de type Pop-up. Pour utiliser CoLab, il faut impérativement autoriser les Pop-up au moins sur le site correspondant à l'URL servant de point d'entrée à CoLab.

V.2.4.2. Sélection et entrée dans une session de navigation coopérative

Dès que le serveur Proxy reçoit une requête d'un utilisateur, il vérifie si l'ordinateur depuis lequel cet utilisateur essaie de se connecter appartient déjà à une session de

navigation coopérative disponible sur le serveur Proxy. Cette vérification est nécessaire puisque notre gestion de session est liée au nom (ou numéro IP) des machines. Nous ne pouvons pas ainsi utiliser le système de gestion de session propre à HTTP parce que, pour des raisons de sécurité, une session HTTP est définie dans le contexte d'un seul serveur. Par exemple, si un utilisateur établit une session HTTP sur l'URL http://www.a.com, et qu'il charge ensuite une page Web hébergée sur le site http://www.b.com, la session établie avec le premier site tombe automatiquement, et il n'y a plus moyen de la récupérer.

Lorsque le serveur Proxy identifie que la machine depuis laquelle un utilisateur essaie de se connecter ne fait pas déjà partie d'une session, il propose à l'utilisateur de se connecter à une session existante (capture d'écran de la *Figure 133*), et ensuite de s'identifier et de s'authentifier (capture d'écran de la *Figure 134*).

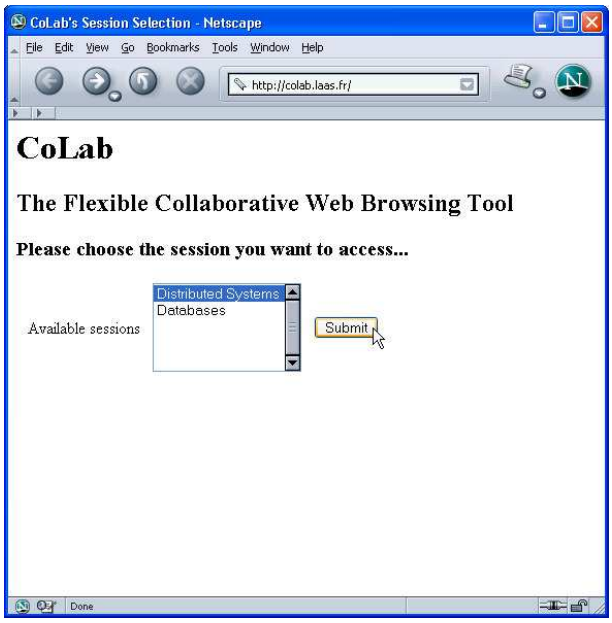


Figure 133. Écran de sélection d'une session CoLab

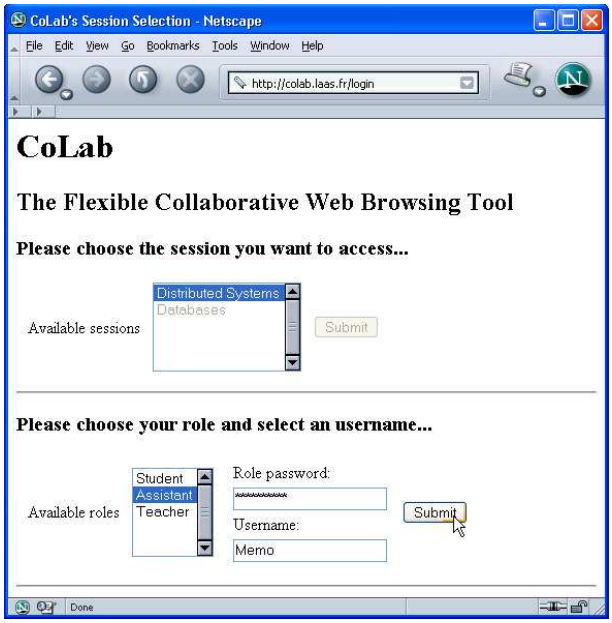


Figure 134. Identification et authentification d'un utilisateur

Une fois l'utilisateur correctement identifié et authentifié, et donc admis à s'enregistrer dans la session qu'il a choisi, une nouvelle fenêtre s'ouvre qui constitue l'interface utilisateur de CoLab. La fenêtre préalablement utilisée pour accéder à CoLab reste quant à elle ouverte et peut être utilisée en dehors du contexte de la session de navigation coopérative. La *Figure 135* présente cette dernière fenêtre.

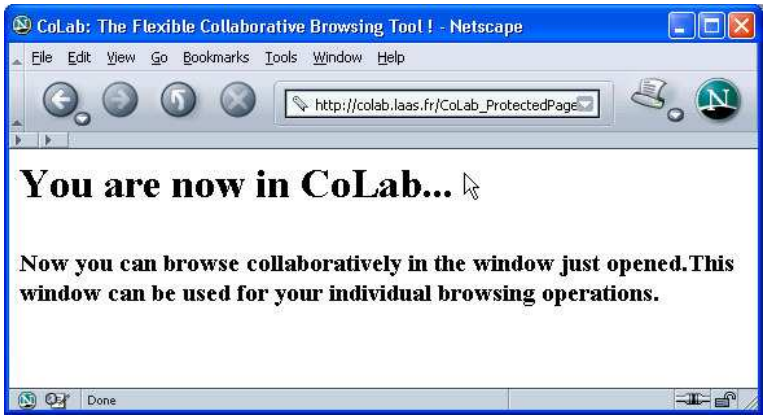


Figure 135. Fenêtre de navigation indépendante de CoLab

V.2.4.3. L'interface utilisateur de CoLab

La fenêtre dans laquelle l'environnement de navigation coopérative est présenté contient une structure de cadres. Le cadre supérieur contient l'interface utilisateur permettant de contrôler CoLab. Le cadre inférieur constitue l'espace de navigation coopérative où sont affichées les pages Web pour cet utilisateur (capture d'écran dans la *Figure 136*).

Figure 136. Fenêtre de CoLab

L'interface graphique utilisateur de CoLab comprend deux composant principaux. Le premier est associé aux boutons de contrôle de la navigation qui restent similaires à ceux d'un navigateur traditionnel. La seule différence notable est que le bouton « HOME » permet de charger la page Web définie, dans le fichier de configuration, comme page par défaut de la session. Le second est associé aux boutons de contrôle de la synchronisation. Cette interface utilisateur est présentée dans la *Figure 137* ci-dessous.

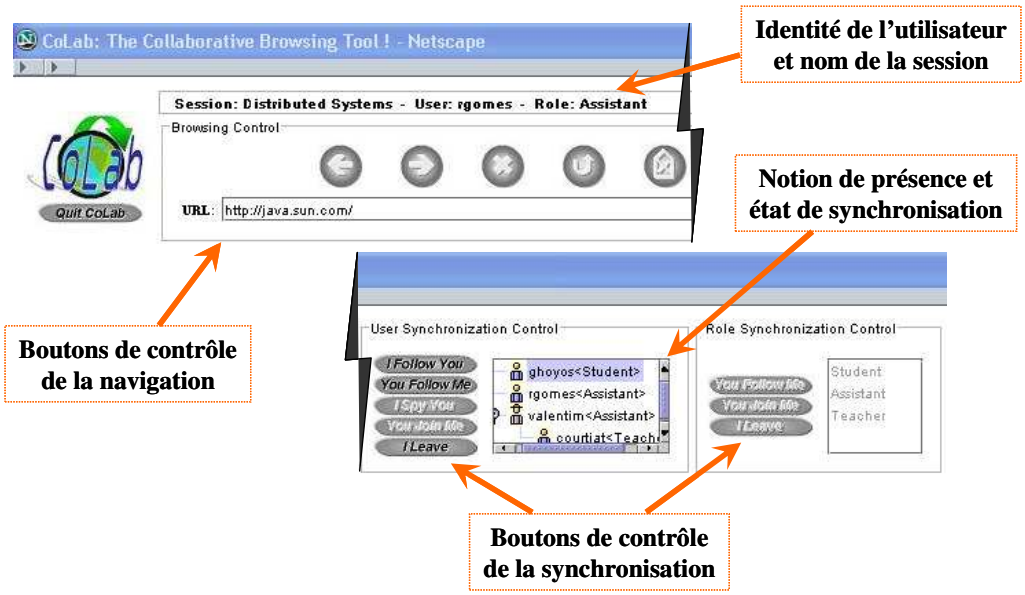


Figure 137. Détail de la GUI de CoLab

Dans la partie inférieure de la figure, en plus des boutons pour contrôler la synchronisation des utilisateurs, nous pouvons voir un affichage des utilisateurs présents dans la session et de leurs relations de synchronisation, ceci sous forme arborescente. Cet affichage permet à chaque utilisateur de connaître à tout instant quels sont les autres utilisateurs connectés à la session, le rôle qu'ils jouent, et les relations de synchronisation établies entre eux.

Nous pouvons par exemple voir dans la *Figure 137* que quatre utilisateurs sont connectés à la session: « ghoyos » avec le rôle de « Student », « rgomes » avec le rôle de « Assistant », « valentim » avec le rôle de « Assistant » et « courtiat » avec le rôle de « Teacher ». Nous pouvons constater que les trois premiers utilisateurs travaillent en mode asynchrone, et que le dernier est synchronisé à l'utilisateur « valentim ».

V.2.4.4. Synchronisation des utilisateurs

Plusieurs boutons implémentent les primitives de synchronisation. Les requêtes de synchronisation, implémentées à l'heure actuelle, sont « I_Follow_You », « You_Follow_Me » et « I_Leave_You » pour des utilisateurs individuels. Cependant, comme nous l'avons déjà dit, ni les requêtes « I_Spy_You » et « You_Join_Me », ni les requêtes groupées ne sont, à l'heure actuelle, implémentées, donc il est naturel que les boutons pour ces requêtes soient grisés.

Les boutons non grisés (qui correspondent aux requêtes de synchronisation disponibles de CoLab) sont validés ou invalidés selon l'état courant de l'utilisateur dans la session. Ainsi, par exemple, le bouton « I_Leave_You » ne sera disponible que, lorsque l'utilisateur qui veut en faire usage, choisit un utilisateur avec lequel il dispose d'une relation de synchronisation. Ce même principe s'applique à tous les boutons.

Lorsqu'un utilisateur, disons « Memo », essaie d'effectuer la requête « I_Follow_You » avec un autre utilisateur, disons « ghoyos », cette requête est envoyée au serveur Proxy. Une fois que le serveur Proxy a vérifié que cette requête ne peut rendre incohérent l'état global de la session, il envoie un message au navigateur de l'utilisateur concerné, sur l'écran duquel apparaît un dialogue lui demandant sont accord. Sur l'écran de l'utilisateur qui a initié la requête apparaît également dialogue lui permettant d'annuler sa requête. Ce mécanisme est illustré dans la *Figure 138*.

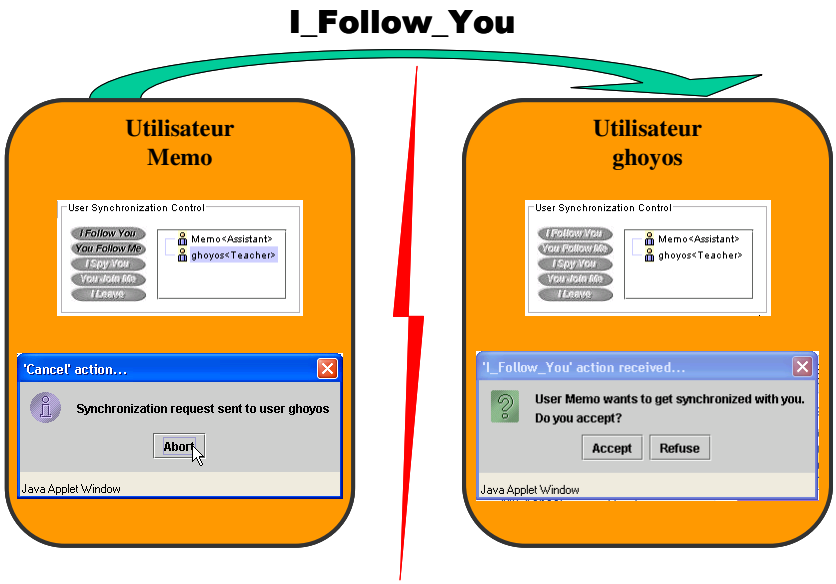


Figure 138. Utilisation de l'action de synchronisation « I_Follow_You »

Si une requête de création de relation de synchronisation est acceptée, l'affichage de l'état de synchronisation est actualisé pour tous les utilisateurs connectés à la session (*Figure 139*). Dans le cas contraire, il n'y a aucune modification de l'affichage, sauf que les dialogues des deux utilisateurs sont effacés.

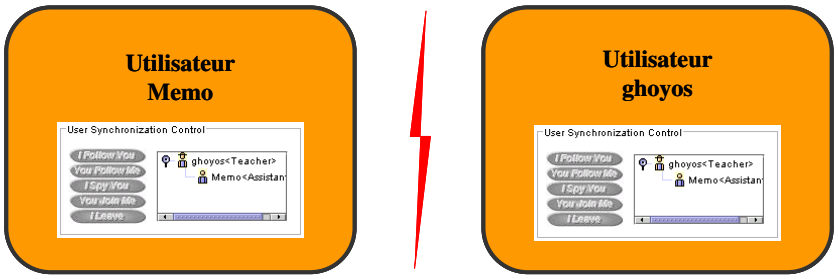


Figure 139. Résultat de l'acceptation de la création d'une relation de synchronisation

Dans le cas de « You_Follow_Me », l'utilisateur qui a initié la requête n'a pas moyen de l'annuler et peut continuer à naviguer librement. Par contre, sur l'écran de l'utilisateur quoi a reçu l'invitation à se synchroniser, apparaît un Pop-up lui permettant d'accepter ou de refuser cette invitation, comme cela est illustré dans la *Figure 140*.

Notons finalement que la troisième et dernière primitive de synchronisation implémentée, « I_Leave_You », est inconditionnelle, ce qui signifie que n'importe quel utilisateur concerné par une relation de synchronisation peut l'utiliser.

Il suffit donc pour un utilisateur de choisir le nom de l'utilisateur qui est synchronisé avec lui, ou avec lequel il est synchronisé, et de cliquer sur le bouton « I_Leave_You » pour que la relation de synchronisation soit supprimée. L'utilisateur qui était jusqu'à maintenant synchronisé devient alors un utilisateur asynchrone vis à vis du système de navigation coopérative.

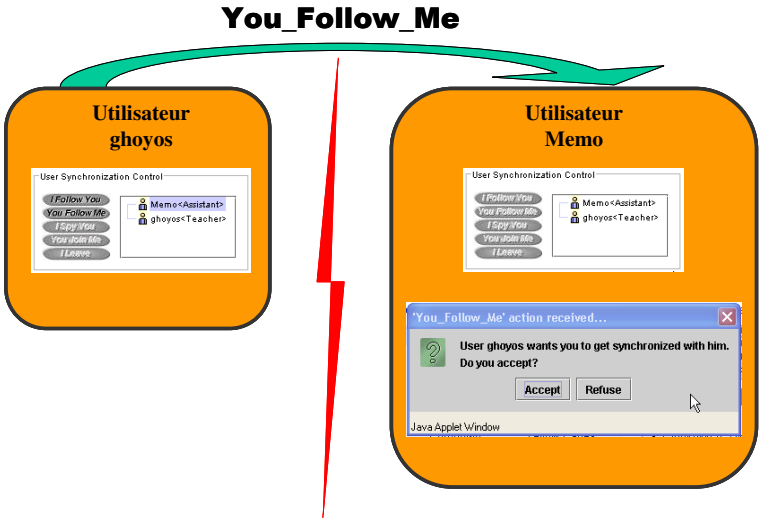


Figure 140. Utilisation de l'action de synchronisation « You_Follow_Me »

V.3. Evaluation des performances de CoLab

Le mécanisme de synchronisation de l'affichage des pages Web est, dans la version courante de CoLab, centralisé sur un serveur Proxy, d'où le risque de créer des goulets d'étranglement à ce niveau, en fonction par exemple, du nombre de sessions disponibles, du nombre d'utilisateurs par session et de la taille des ressources Web rapatriées.

Afin d'évaluer les performances de CoLab nous allons, dans un premier temps, évaluer le coût pour synchroniser l'affichage de plusieurs pages puis, dans un deuxième temps, présenter un ensemble de mesures réalisées avec CoLab.

V.3.1. Evaluation du coût de la synchronisation

Nous allons évaluer le coût pour afficher de manière synchronisée la même page Web sur un ensemble d'utilisateurs synchronisés (ces utilisateurs appartiennent par définition au même SDT). Dans ce but, nous identifions les coûts élémentaires suivants qui représentent du temps (ils pourraient par exemple être exprimés en milli-secondes):

- $C_{Request}$: coût de traitement d'une requête HTTP par le serveur Proxy;
- $C_{Retrieve}$: coût de rapatriement d'une ressource à partir du serveur d'origine ;
- $C_{RetrieveCache}$: coût de rapatriement d'une ressource à partir du cache (serveur Proxy)
- $C_{Translate}$: coût de traduction d'une ressource HTML ;
- $C_{StoreCache}$: coût de stockage d'une ressource dans le cache du serveur Proxy ;
- $C_{Transmit}$: coût de rapatriement d'une ressource du serveur Proxy vers un utilisateur.

Partant de ces coûts individuels nous pouvons calculer le coût pour synchroniser l'affichage d'une page Web élémentaire (i.e. ne contenant pas de ressources internes) sur N utilisateurs (i.e. le SDT associé comprend N utilisateurs, et l'utilisateur associé à la racine de cet SDT a sélectionné, par une action de navigation, la page à afficher).

$$C_{SDT} = (C_{Request} + C_{Retrieve} + C_{Translate} + C_{StoreCache} + C_{Transmit}) + (N - 1) * (C_{Request} + C_{RetrieveCache} + C_{Transmit})$$

V.3.2. Mesures réalisées sur CoLab

Dans le but de vérifier la capacité de réponse de notre plate-forme expérimentale, nous avons exécuté une série de tests avec un nombre d'utilisateurs compris entre 1 et 165 qui peuvent visiter jusqu'à 20 sites Web différents. Les sites utilisés pour réaliser les tests ont été choisis de manière à être représentatifs pour ce qui concerne la taille et la complexité des pages Web rapatriées.

Ces tests ont été réalisés en utilisant des ordinateurs non dédiés à la plate-forme et raccordés sur le réseau du LAAS-CNRS. À partir de ces tests, nous avons effectué un analyse comparative du temps de rapatriement des ressources Web en fonction du nombre d'utilisateurs connectés à une même session de navigation coopérative gérée par un serveur Proxy centralisé.

Les sites Web choisis ont été les suivants :

- http://jakarta.apache.org/tomcat/
- http://jakarta.apache.org/tomcat/tomcat-3.3-doc/index.html
- http://jakarta.apache.org/tomcat/tomcat-3.3-doc/readme
- http://java.sun.com/
- http://java.sun.com/downloads/index.html
- http://java.sun.com/j2se/1.5.0/docs/api/
- http://java.sun.com/reference/api/index.html
- http://www.acm.org/
- http://www.acm.org/pubs/
- http://www.apache.org/
- http://www.cnrs.fr/
- http://www.google.fr/
- http://www.google.fr/search?btnG=Recherche+Google&hl=fr&ie=ISO-8859-1&meta=&q=xalapa
- http://www.mit.edu/
- http://www.mit.edu/education/
- http://www.w3.org/
- http://www.w3.org/MarkUp/
- http://www.w3.org/TR/html4/
- http://www.xalapa.gob.mx/
- http://www.xalapa.gob.mx/mensaje_esp.htm

Nous présentons, dans la *Figure 141*, le temps moyen de rapatriement des ressources pour chaque groupe comprenant respectivement 1, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150 et 165 utilisateurs (en fait des applications clientes en Java) qui sont tous synchronisés à un même utilisateur qui dirige la navigation (en l'occurrence un opérateur humain), et ceci en fonction de la taille moyenne de la ressource rapatriée (de 1 783 octets à 36 461 octets).

Nous remarquons que le temps moyen de rapatriement est, en général, proportionnel au nombre d'utilisateurs connectés à la session et à la taille de la ressources Web rapatriée, ce qui semble tout à fait normal. Certains pics dans la figure correspondent à des anomalies, qui sont probablement produites non pas par le serveur Proxy, mais par la machine hébergeant l'application. Rappelons que ces tests ont été réalisés sur des machines non dédiées, qui, peuvent à un instant donné, exécuter une autre application. Nous remarquons également que le temps de rapatriement est très faible (de quelques milli-secondes à quelques dizaines de milli-secondes) pour de petits groupes d'utilisateurs synchrones.

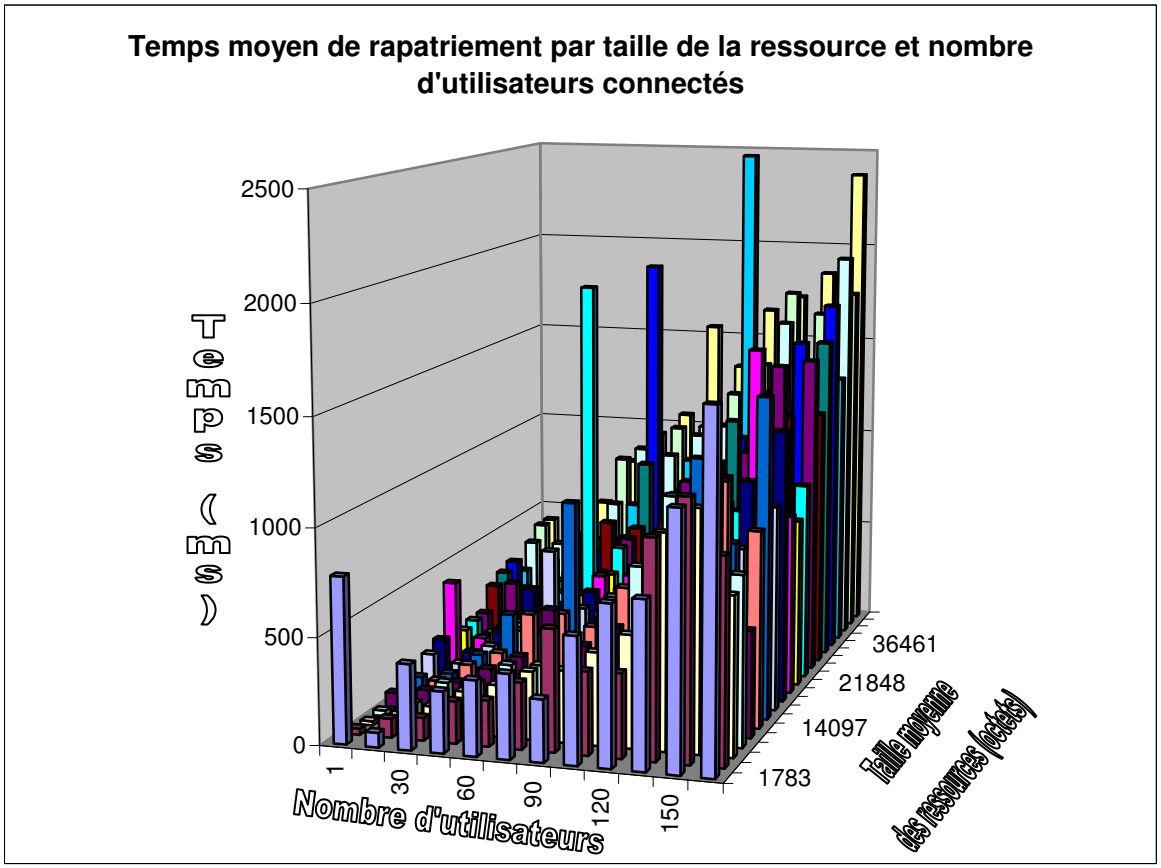


Figure 141. Statistiques sur le temps moyen de rapatriement selon la taille des ressources et le nombre d'utilisateurs

Nous présentons dans la *Figure 142* les mêmes résultats mais sous une forme différente. Nous faisons ici la moyenne des temps de rapatriement pour chaque groupe d'utilisateurs, sachant que la taille moyenne de l'ensemble des ressources Web rapatriées est égale à 22 587 octets. Là encore, nous constatons l'augmentation du temps moyen de rapatriement d'une page Web en fonction du nombre d'utilisateurs connectés à la session. Nous remarquons que le temps de rapatriement reste inférieur à 400 milli-secondes pour des groupes de taille inférieure à 50 utilisateurs. La progression semble linéaire au moins jusqu'à 165 utilisateurs.

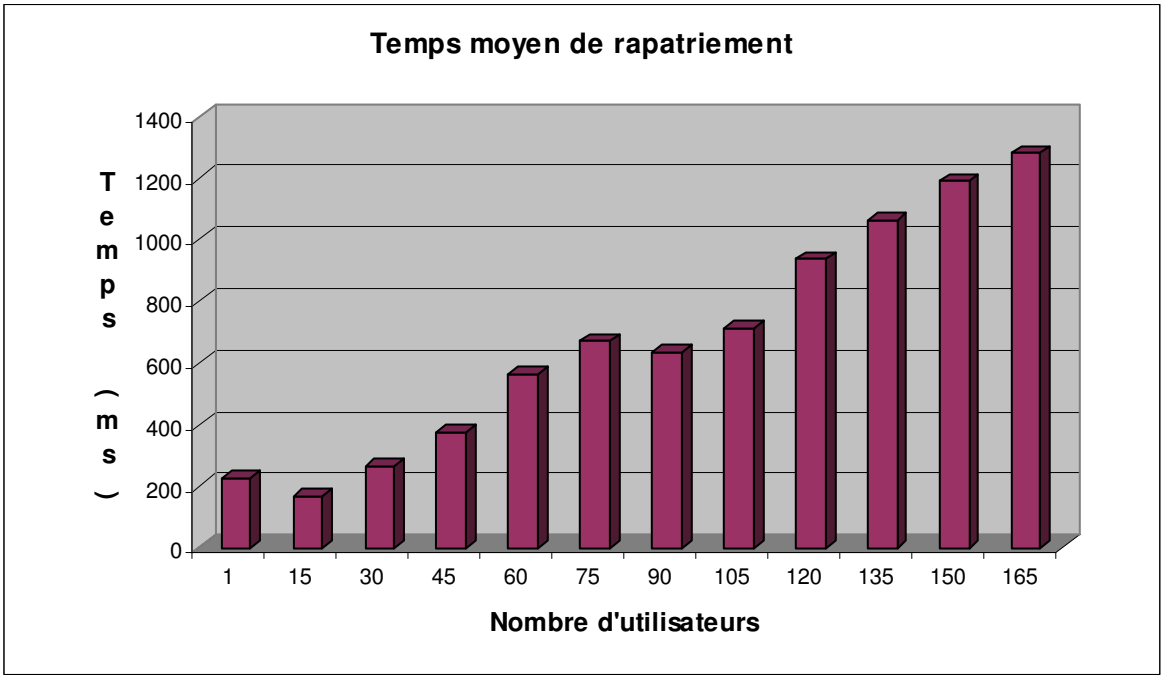


Figure 142. Statistiques sur le temps moyen de rapatriement selon le nombre d'utilisateurs

V.4. Mise en œuvre de CoLab en tant que service Web

Le but principal de la technologie des services Web est d'offrir un moyen pour déployer et mettre à disposition des services qui pourront être accédés par des applications client de manière automatique, contrairement au mode de fonctionnement actuel où ce sont des utilisateurs humains qui accèdent directement aux applications mises à disposition sur le Web. Ces idées sont illustrées dans la *Figure 143*.

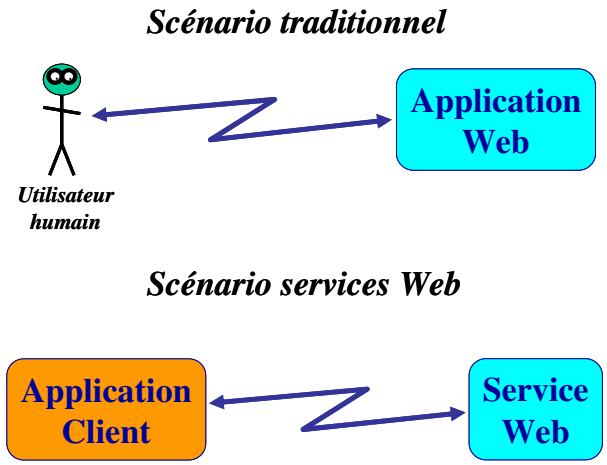


Figure 143. Le paradigme des services Web

Dans cette figure l'application Web peut prendre le rôle de l'application client, et c'est à travers cette application qu'un utilisateur peut avoir alors accès au service Web.

La technologie des services Web est largement reconnue notamment pour ses capacités d'interopérabilité, étant donné qu'elle permet à des applications de se communiquer directement grâce à des protocoles comme HTTP ou SMTP (Simple Mail Transport Protocol – protocole simple de transport de courrier). Elle s'appuie sur trois

standards principaux : WSDL (Web Services Description Language – langage de description de services Web), qui permet la description des caractéristiques des services Web ; UDDI (Universal Description, Discovery and Integration – description, découverte et intégration universelle), qui implémente un mécanisme de découverte des services Web disponibles sur Internet ; et SOAP (Simple Object Access Protocol – protocole simple d'accès aux objets), qui permet de faire des appels à des services Web disponibles. L'utilisation de ces standards s'appuie sur un autre standard : XML (eXtensible Markup Language – langage extensible de marquage) pour le transfert d'information.

Dans ce contexte, nous montrons dans la *Figure 144* comment tous ces composants interagissent. Nous constatons que le fournisseur de service publie son service Web avec sa description. Le demandeur de service accède à l'agent de découverte, qui lui envoie en réponse la localisation du service et sa description.

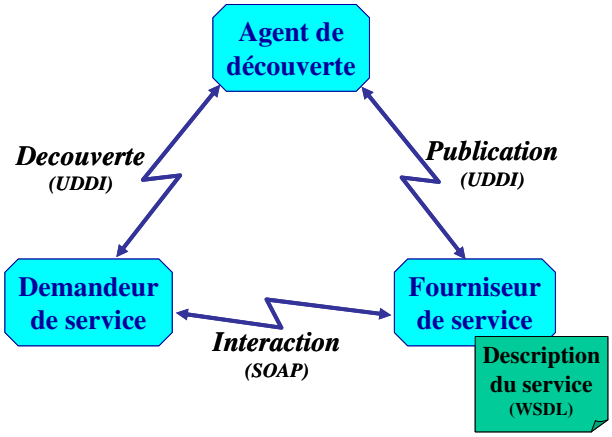


Figure 144. Schéma d'interactions pour des services Web

L'objectif derrière une mise à disposition de CoLab en tant que service Web est de permettre à d'autres applications d'interagir avec cette plate-forme de navigation coopérative sans aucune intervention humaine. Ainsi, une application ayant besoin d'utiliser des services de navigation coopérative sur le Web pourra créer et configurer des sessions pour qu'elles soient rendues disponibles aux utilisateurs de cette application.

Nous avons ainsi développé une application Web à travers laquelle il est possible de définir des sessions de navigation coopérative. Lorsqu'un utilisateur crée une session de navigation coopérative, il reçoit un formulaire où il peut dans un premier temps décrire la session, ce qui inclut, comme nous l'avons déjà vu, la spécification du nom de la session, de l'URL initial, des rôles disponibles et des mots de passe associés. L'interface associée est illustrée dans la *Figure 145*.

Dans un deuxième temps, lorsque l'utilisateur clique sur le bouton OK, l'application présente un nouveau formulaire pour spécifier les privilèges existants entre les rôles de la session. Cette interface est présentée dans la *Figure 146*.

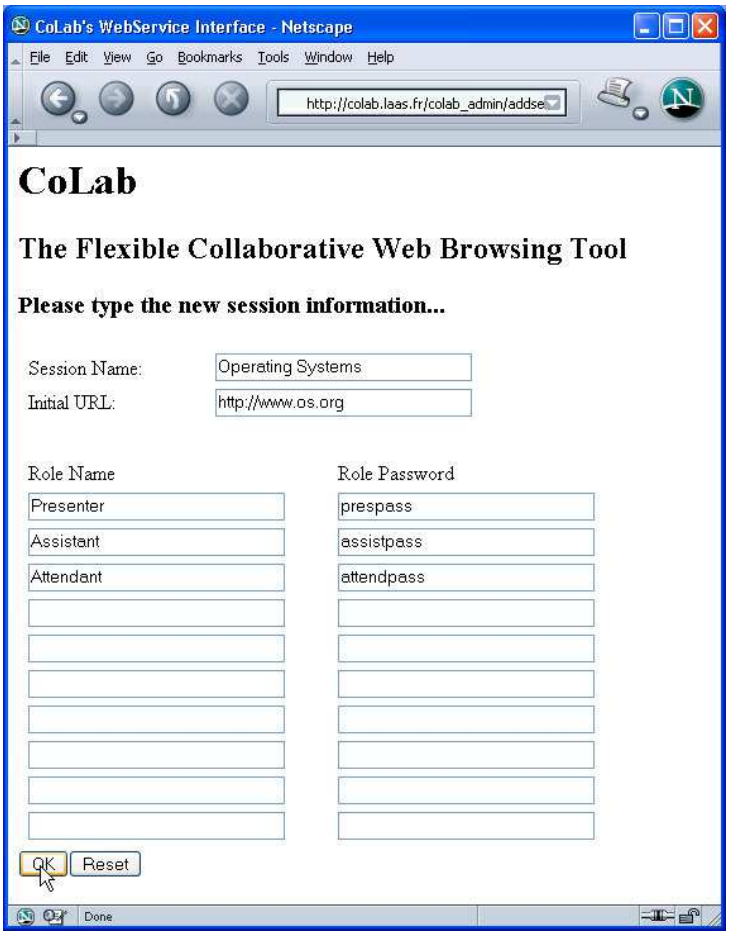


Figure 145. Interface de définition des paramètres d'une session

Cette information est alors envoyée au serveur Proxy au moyen du protocole SOAP ; le serveur Proxy vérifie si la session demandée peut être effectivement créée. Dans l'affirmative, l'utilisateur reçoit en réponse l'information nécessaire (URL du fichier PAC et URL d'accès à la plate-forme) pour accéder à la session de navigation coopérative qui vient d'être créée. Il peut soit l'utiliser lui-même en configurant son navigateur, soit la passer à d'autres personnes.

Nous avons testé cette application et avons pu valider que la mise à disposition de CoLab en tant que service Web marche effectivement. Il ne reste plus qu'à publier ce service Web en utilisant UDDI pour que d'autres applications puissent créer des sessions de navigation coopérative et les mettre à disposition d'autres personnes.

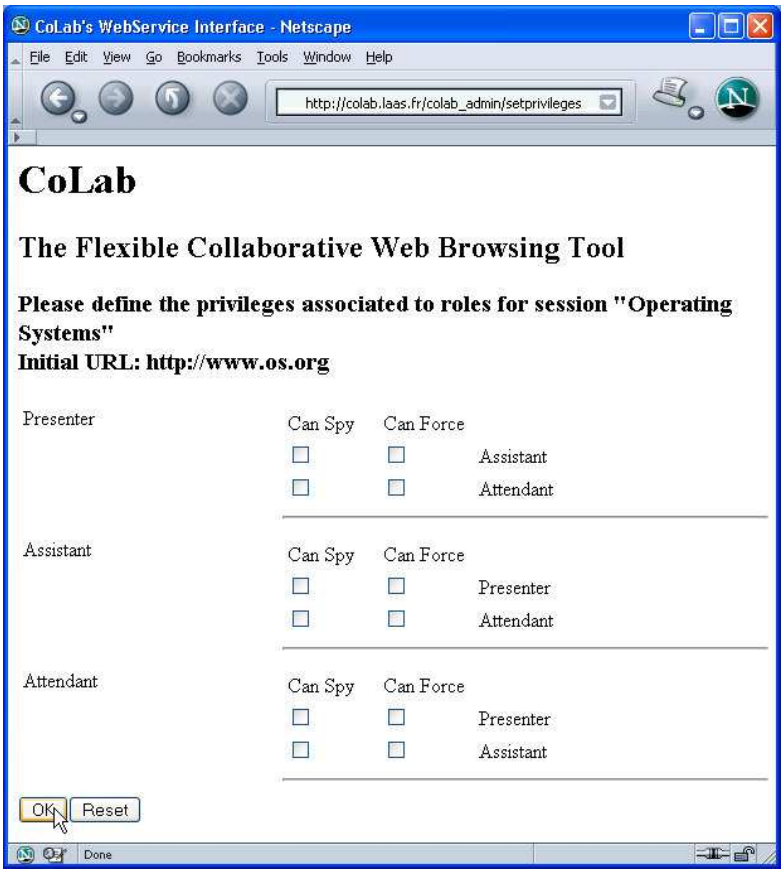


Figure 146. Définition des privilèges entre les rôles définis

V.5. Evolution vers une architecture répartie comprenant plusieurs serveurs Proxy

Nous envisageons dans ce paragraphe de faire évoluer l'architecture proposée pour être en mesure d'équilibrer la charge de traitement des requêtes de navigation sur plusieurs serveurs Proxy.

Par définition, les actions de navigation exécutées au sein d'un SDT sont complètement indépendantes des actions de navigation exécutées dans d'autres SDTs. L'idée de base de notre proposition consiste par conséquent à allouer la gestion de différents SDTs à différents serveurs Proxy avec comme seule contrainte que :

Un serveur Proxy pourra gérer plusieurs SDTs de la même session ou de sessions différentes, mais un SDT ne pourra être géré que par un seul serveur Proxy.

La décision de répartir des SDTs sur plusieurs serveurs prend en compte le fait que le coût nécessaire pour rapatrier une ressource est, en principe, beaucoup moins important pour un utilisateur synchrone que pour un utilisateur asynchrone. Un utilisateur synchrone récupère toujours sa ressource du cache implémenté dans un Proxy, alors qu'un utilisateur asynchrone récupère en général sa ressource d'un serveur Web distant.

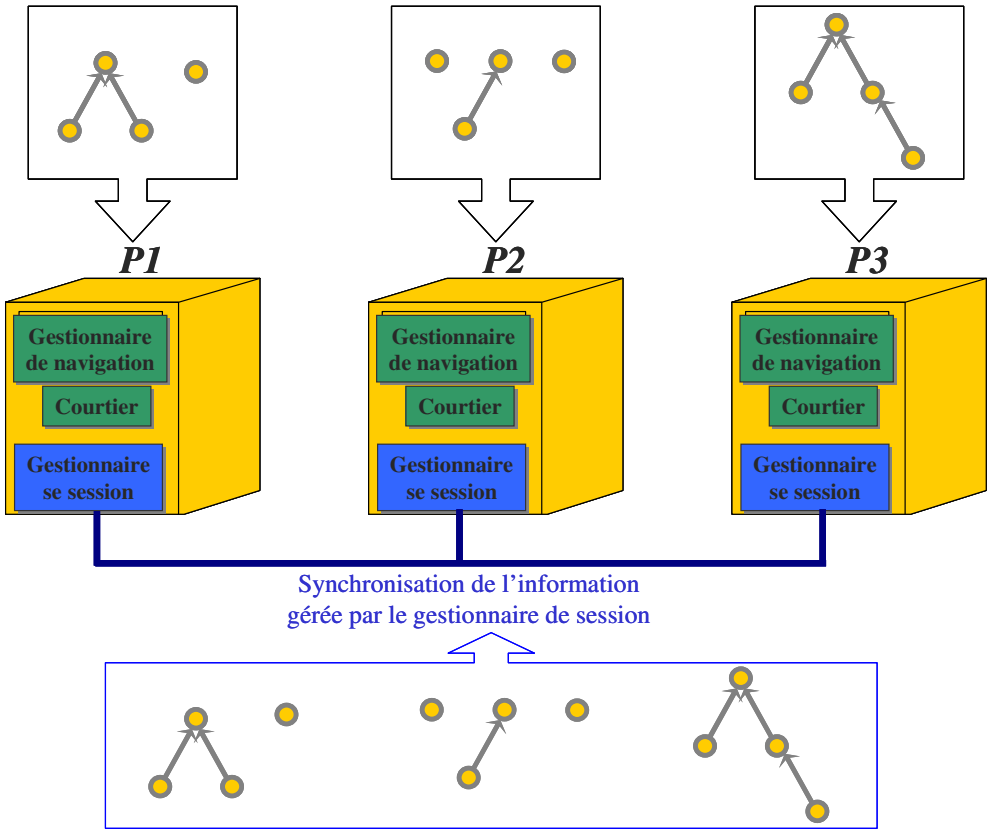


Figure 147. Evolution de CoLab vers une architecture répartie

L'architecture proposée est illustrée dans la *Figure 147*, où nous définissons trois serveurs Proxy, appelés P1, P2 et P3. Chaque Proxy implémente les fonctionnalités du serveur Proxy centralisé, et donc en particulier les services de gestionnaire de session, de gestionnaire de navigation et de courtier. Nous faisons l'hypothèse que l'information gérée par les gestionnaires de session des différents serveurs Proxy est synchronisée grâce à la mise en œuvre d'un protocole ad-hoc qui ne sera pas détaillé ici; en conséquence chaque serveur Proxy a une vue globale de l'ensemble des sessions disponibles et de l'état courant de synchronisation des utilisateurs (SDTs) au sein des sessions.

Différents critères peuvent être envisagés pour décider quand et comment migrer un SDT d'un serveur Proxy à un autre, comme par exemple :

- le nombre de SDTs gérés par les différents serveurs Proxy ;
- la complexité des structures de synchronisation des différents SDTs dans les différents serveurs Proxy ;
- des priorités (à définir) entre différentes session, ou des rôles des utilisateurs dans une session.

L'idée est ainsi de concevoir un mécanisme permettant d'adapter de manière automatique la charge des différents Proxy, sachant qu'il ne faudra pas négliger le coût lié à la migration d'un SDT d'un serveur Proxy à un autre.

L'accès des navigateurs aux différents serveurs Proxy répartis peut s'appuyer sur la même technique de traduction des pages Web que nous avons définie dans le Chapitre V. Cette technique se généralise avec l'inclusion d'un nouveau paramètre « COLAB_PROXY » dont la valeur contient le nom du serveur Proxy vers lequel devra être envoyée la requête HTTP suite à un clic utilisateur. Ceci implique la modification de la structure du fichier PAC comme cela est illustré dans la *Figure 148*.

```
function findProxyForURL(url, host) {
       if(host.toLowerCase() == "colab.laas.fr")
              return "PROXY banderilla.laas.fr:8090";
       else if(url.toUpperCase().indexOf("COLAB_PROXY=") >= 0)
              return "PROXY " +
                     url.substring(
                            url.toUpperCase().indexOf("COLAB_PROXY=")) +
                     ":8090";
       else
              return "DIRECT";
}
```

Figure 148. Fichier de configuration automatique de serveur Proxy pour le cas reparti

V.6. Conclusion

Dans ce chapitre, nous avons présenté en détail l'implémentation, l'utilisation et le déploiement de l'outil CoLab qui constitue à ce jour une plate-forme de navigation coopérative opérationnelle s'appuyant sur un serveur Proxy centralisé.

Nous avons évalué les performances de CoLab et avons montré que les performances obtenues sont tout à fait satisfaisantes pour des groupes de taille raisonnable (plusieurs dizaines d'utilisateurs). Nous avons également montré que le service de navigation coopérative assuré par Colab pouvait facilement être mis à disposition sous la forme d'un service Web et avons, dans ce but, développé une application simple de création de sessions de navigation coopérative.

Finalement, nous avons envisagé de faire évoluer l'architecture de CoLab en la distribuant sur plusieurs Proxy, et le SDT nous a paru représenter la structure de donnée adéquate pour décider comment allouer les utilisateurs aux différents serveurs Proxy.

Chapitre VI

Conclusion générale

Dans ce mémoire de thèse nous avons présenté la conception, la formalisation, l'implémentation, le déploiement et l'évaluation de performances d'un système de navigation coopérative sur le Web. Ce système de navigation coopérative est complètement original aussi bien dans ses principes (modèle de synchronisation utilisé), dans son implémentation (technique de traduction des pages HTML) que dans son déploiement (proposition de rendre disponible le service de navigation coopérative sous la forme d'un service Web).

Parmi les différentes contributions que nous apportées dans ce travail, trois, sur lesquelles nous allons revenir, nous paraissent essentielles.

Nous avons défini un modèle de synchronisation qui permet de manière intuitive et simple de créer des relations de synchronisation entre utilisateurs appartenant à une même session de navigation coopérative. Les requêtes « I_Follow_You », « You_Follow_Me » et « I_Leave_You », en association avec la requête de navigation, constituent le modèle de synchronisation de base qui a été formalisé et vérifié dans différentes configurations. L'approche de spécification utilisée nous a permis de facilement engendrer les réseaux de Petri associés aux configurations intéressantes qui pouvaient être traités par les outils TINA et ALDEBARAN, et ainsi de mieux maîtriser en pratique l'explosion combinatoire de l'espace d'états. Nous avons montré également comment étendre le modèle de synchronisation de base en introduisant de nouvelles requêtes de synchronisation (« You_Join_Me » et « I_Spy_You ») paramétrées par les rôles des utilisateurs. Nous avons formalisé ce modèle étendu mais ne l'avons pas vérifié à cause des problèmes d'explosion combinatoire.

Partant des modèles de synchronisation, nous avons défini formellement l'architecture du service de navigation coopérative, en utilisant le profil UML/SDL supporté par l'outil TAU G2 de Telelogic. Nous avons défini une méthode de spécification de l'architecture s'appuyant 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. Cette modélisation UML a servi de pont entre les modèles de synchronisation abstraits définis dans le chapitre III et l'implémentation présentée dans le chapitre V. Notons en particulier que le SDT (*Synchronization Dependency Tree*), l'arbre de synchronisation représentant les relations de synchronisation, s'est avéré particulièrement bien adapté tant pour la modélisation et la vérification que pour l'implémentation.

Notre troisième contribution importante est associée à l'implémentation de l'outil CoLab. Nous avons justifié et développé une technique originale de traduction des pages Web qui s'est avérée très flexible, sans être limitative du point des performances comme

l'ont montré les mesures effectuées sur CoLab. Nous avons implémenté l'architecture dans un premier temps sur un serveur Proxy centralisé et donné des pistes pour la mettre en œuvre de manière répartie sur plusieurs serveurs Proxy.

Plusieurs perspectives s'offrent à nous pour l'approfondissement de ce travail.

La première perspective consiste, sur les bases que nous avons indiquées, à développer une architecture répartie de CoLab sur plusieurs serveurs Proxy. L'objectif essentiel serait ici de définir des protocoles permettant de synchroniser la connaissance de l'état global des sessions (qui est synchronisé à qui dans quelle session ?) et de faire migrer des arbres de synchronisation (SDT) d'un serveur Proxy à un autre. Il serait également nécessaire de définir des règles politiques, pouvant dépendre de nombreux critères, pour décider quand migrer ces arbres de synchronisation. Finalement, nous pourrions également envisager des structures de cache Web réparties, domaine dans lequel de nombreux travaux existent déjà comme dans [Fan-98] [Gwertzman-96] [Holmedahl-98] [Aggarwa-99].

Une deuxième perspective consiste à lever certaines limitations de Colab. Un premier objectif serait d'implémenter toutes les fonctionnalités définies dans le modèle de synchronisation étendu, en particulier celles qui dépendent des rôles des utilisateurs et des privilèges qu'un rôle peut avoir sur un autre. Un deuxième objectif consisterait à lever la contrainte comme quoi toutes les ressources traitées doivent être au format HTML, une évolution vers des documents XML serait probablement intéressante et facile à implémenter. Un troisième objectif serait de lever quelques limitations techniques qui n'ont pas été résolues à ce jour. CoLab n'est ainsi pas en mesure de traiter automatiquement des pages contenant des hyperliens définis au moyen de scripts (par exemple, en JavaScript ou JScript), des pages contenant des hyperliens codés en binaire (par exemple via la technologie Flash) ou des pages contenant du code JSP. Une solution est envisageable pour la première contrainte (scripts) mais les deux contraintes suivantes semblent plus difficiles à résoudre.

Une troisième perspective consiste à généraliser le modèle de synchronisation à des médias continus (par exemple des flux audio et/ou vidéo). Une première expérience dans ce sens, qui n'a pas été décrite dans le manuscrit, a fait l'objet d'une première publication [Hoyos-05b]. Ce travail sera poursuivi et approfondi.

Une dernière perspective consiste à définir des APIs pour intégrer plus facilement CoLab à d'autres applications coopératives. Un travail dans ce sens a été initié dans le cadre de la définition et de l'implémentation d'un environnement d'intégration d'applications coopératives, appelé LEICA [Gomes-05c].

Plusieurs scénarios d'application peuvent être envisagées pour notre plate-forme de navigation coopérative sur le Web. Celle qui semble la plus appropriée, mais qui n'est pas la seule, est l'implémentation des systèmes orientés à l'éducation [Ausserhofer-99] [Specht-00] [Sanrach-00]. Dans des tels scénarios on peut facilement imaginer des utilisateurs qui prennent des rôles de professeur, d'assistant de professeur, et des élèves. Des privilèges peuvent être alors associés à ces rôles, et des dynamiques de coopération peuvent être mises en place grâce à la possibilité de créer et de détruire des relations de synchronisation entre les utilisateurs.

Références bibliographiques

Publications de l'auteur

[Gomes-05c] Lima-Gomes, R.; Hoyos-Rivera, G.J. & Courtiat, J.P.
"LEICA: Loosely-coupled Environment for Integrating Collaborative Applications"
16th International Conference on Database and Expert System Applications (DEXA 2005)- Web Based Collaboration (WBC 2005)
Copenhagen, Denmark, 22-26 août, 2005

[Gomes-05b] Lima-Gomes, R.; Hoyos-Rivera, G.J. & Courtiat, J.P.
"Integrating Collaborative Applications with LEICA"
3rd IEEE International Conference on Information Technology:
Research and Education (ITRE 2005)
Hsinchu, Taiwan, 27-30 Juin, 2005

[Gomes-05a] Lima-Gomes, R.; 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) (Springer Verlag)
Athens, Greece, 15-17 Juin 2005

[Hoyos-05b] Hoyos-Rivera, G.J.; Lima-Gomes, R.; Courtiat, J.P. & Wilrich R.
"Collaborative Web Browsing Tool Supporting Audio/Video Interactive Presentations"
IEEE 14th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE'2005)
Linköping (Suède), 13-15 juin, 2005.

[Hoyos-05a] Hoyos-Rivera, G.J.; Lima-Gomes, R. & Courtiat, J.P.
"CoLab: A Flexible Collaborative Web Browsing Tool"
IEEE 19th International Conference on Advanced Information Networking and Applications (AINA'05)
Taipei (Taiwan), 28-30 mars, 2005.

[Gomes-03b] Lima-Gomes, R.; Hoyos-Rivera, G.J. & Courtiat, J.P.
"Regarding the Integration of Collaborative Applications Into Virtual Worlds."
Cooperative Information Systems International Conference (CoopIS'03)
Catanie (Italie), 3-7 Novembre 2003

[Hoyos-03] Hoyos-Rivera, G.J.; Lima-Gomes, R.; 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)
Athènes (Grèce), 9-11 Juillet 2003

[Baudin-03] Baudin, V.; Courtiat, J.P.; Lima-Gomes, R.; Hoyos-Rivera, G.J. & Villemur, T.
"An e-Learning Collaborative Platform for Laboratory Education"
4th International Conference on Information Technology Based Higher Education and Training (ITHET'03)
Marrakech (Maroc), 7-9 Juillet 2003

[Gomes-03a] Lima-Gomes, R.; Hoyos-Rivera, G.J. & Courtiat, J.P.
"Collaborative Virtual Environments: Going Beyond Virtual Reality"
2003 IEEE International Conference on Multimedia (ICME'2003)
Baltimore (USA), 6-9 Juillet 2003

[Gomes-02] Lima-Gomes, R.; Hoyos-Rivera, G.J. & Courtiat, J.P.
"A Flexible Architecture for Collaborative Browsing"
International Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises (WETICE'2002)
Pittsburgh (USA), 10-12 Juin 2002

[Hoyos-02] Hoyos-Rivera, G.J.
"Une Architecture Flexible de Navigation Coopérative"
3ème Congrès des Doctorants de l'Ecole Doctorale Systèmes
Toulouse (France), 22-23 Mai 2002

[Hoyos-01] Hoyos-Rivera, G.J. ; Courtiat, J.P. & Villemur, T.
"A Design Framework for Collaborative Browsing"
IEEE 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2001)
Cambridge (USA), 20-22 Juin 2001

Bibliographie

[Aggarwal-99] Aggarwal, C. ; Wolf, J.L. & Yu, P.S.
"Caching on the World Wide Web"
IEEE Transactions on Knowledge and Data Engineering
Volume 11, Issue 1 (January 1999)
Pages: 94 - 107
ISSN:1041-4347

[Ausserhofer-99] Ausserhofer, A.
"Web-Based Teaching and Learning: A Panacea?"
IEEE Communications Magazine
Volume: 37, Issue: 3, Pages 92-96
March 1999

[Baudin-04] Baudin, V.; Faust,M.; Kaufmann,H. et Al.
"The Lab@Future Project: 'Moving Towards the Future of e-Learning'"
IFIP TC3 Technology Enhanced Learning Workshop (Tel'04)
World Computer Congress (WCC'04)
August 22-27, 2004, Toulouse, France
Series: IFIP International Federation for Information Processing, Vol. 171
Courtiat, Jean-Pierre; Davarakis, Costas; Villemur, Thierry (Eds.)
2005, XII, 188 p., Hardcover
ISBN: 0-387-24046-2

[Berthomieu-04] Berthomieu, B; Ribet, P.O. & Vernadat, F
"The tool TINA - Construction of abstract state spaces for Petri nets and time Petri nets"
International Journal of Production Research, Vol.42, N° 14, pp.2741-2756
July 15,2004

[Björkander-00a] Björkander, M.
"Graphical Programming Using UML and SDL"
IEEE Computer
Volume 33, Number 12, January 2000

[Björkander-00b] Björkander, M.
"Real-time Systems in UML (and SDL)"
Embedded Systems Engineering Magazine
October/November 2000
http://www.esemagazine.co.uk/

[Björkander-03] Björkander, M. & Kobryn, C.
"Architecting Systems with UML 2.0"
IEEE Software
Volume 20, Number 4, July/August 2003

[Boticario-00] Boticario, J.G,; Gaudioso, E. & Hernández, F.
"Adaptive Navigation Support and Adaptible Collaboration Support in WebDL"
Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000
Trento, Italy, August 28-30, 2000
Lecture Notes in Computer Science 1892 Springer 2000
ISBN 3-540-67910-3

[Bouthors-99] Bouthors, V. & Dedieu, O.
" Pharos, a Collaborative Infraestructure for Web Knowledge Sharing"
ECDL'99: LNCS 1696
Springer-Verlag Berlin Heidelberg, 1999, pp. 215 – 233

[Bullinger-97] Bullinger, H.J.; Warschat, J. & Schumacher, O
"Floor Control in Collaborative Design Environments"
Design of Computing Systems: Cognitive Considerations
7th International Conference on Human-Computer Interaction, (HCI'97)
San Francisco, California, USA, August 24-29, 1997, Volume 1. Elsevier
ISBN 0-444-82183-X

[Cabri-01] Cabri, G.; Leonardi, L. & Zambonelli, F.
"Web-assisted Visits to Cultural Heritage"
10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises: WetIce'2001, Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises
MIT, Cambridge, Massachusetts, USA, 2001.

[Cabri-99] Cabri, G.; Leonardi, L. & Zambonelli, F.
"Supporting Cooperative WWW Browsing: a Proxy-based approach"
7th Euromicro Workshop on PDP, IEEE, 1999

[Chong-00] Chong, S.T. & Sakauchi, M.
"E-CoBROWSE: co-Navigating the Web with Chat-pointers and Add-ins – Problems and Promises"
IASTED ICPDS, Collaborative Technologies Symposium, ACM
Las Vegas, Nevada, USA, Nov. 2000

[Damianou-01] Damianou, N.; Dulay, N,; Lupu, E. et Al.
"The Ponder Policy Specification Language"
Policies for Distributed Systems and Networks
International Workshop, POLICY 2001 Bristol, UK
January 29-31, 2001
Proceedings. Lecture Notes in Computer Science 1995 Springer
ISBN 3-540-41610-2

[Diaz-01] Diaz, M.
"Les Réseaux de Petri. Modèles Fondamentaux"
Hermes Science
Traite IC2 Information-Commande-Communication
ISBN 2-7462-0250-6, 2001, 384p

[Diaz-03] Diaz, M.
"Vérification et mise en œuvre des réseaux de Petri"
Lavoisier, 2003
ISBN: 2-7462-0445-2

[Dommel-97] Dommel, H.P. & Garcia-Luna-Aceves, J.J.
"Floor Control for Multimedia Conferencing and Collaboration"
Cluster Computing Journal
Publisher: Springer-Verlag GmbH
ISSN: 0942-4962 (Paper) 1432-1882 (Online)
Issue: Volume 5, Number 1, January 1997

[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.
ISSN: 1386-7857 (Paper) 1573-7543 (Online)
Issue: Volume 2, Number 1, March 1999

[Duflos-02] Duflos, S.; Diaz, G.; Gay, V. & Horlait, E.
"Comparative Study of Policy Specification Languages for Secure Distributed Applications"
Management Technologies for E-Commerce and E-Business Applications
13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002
Montreal, Canada, October 21-23, 2002
Lecture Notes in Computer Science 2506 Springer
ISBN 3-540-00080-1

[Fan-98] Fan, L.; Cao, P. ; Almeida, J. et Al.
"Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol"
Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication
Vancouver, British Columbia, Canada, 1998
Pages: 254 - 265
ISSN:0146-4833

[Fernandez-96] Fernandez, J.C; Garavel, H.; Kerbrat, A. et Al.
"CADP - A Protocol Validation and Verification Toolbox"
8th International Conference Computer Aided Verification, CAV '96
New Brunswick, NJ, USA, July 31 - August 3, 1996
Lecture Notes in Computer Science 1102 Springer 1996, ISBN 3-540-61474-5

[Gori-00] Gori, M.; Maggini, M.; Martinelli, E. et Al.
"Learning User Profiles in NAUTILUS"
Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000
Trento, Italy, August 28-30, 2000
Lecture Notes in Computer Science 1892 Springer 2000
ISBN 3-540-67910-3

[Gwertzman-96] Gwertzman, J. & Seltzer, M.
"World Wide Web Cache Consistency"
Proceedings of the USENIX 1996 Annual Technical Conference
San Diego, California, January 1996

[Hall-01] Hall, M.
"Core Servlets and JavaServer Pages"
Sun Microsystems Press – A Prentice Hall Title, 2001
ISBN: 0-13-08934904

[Han-00] Han, R.; Perret, V. & Naghshineh, M.
"WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing"
CSCW'00, IEEE
Philadelphia, PA, USA, Dec. 2000

[Henze-00] Henze, N. & Nejdl, W.
"Extendible Adaptive Hypermedia Courseware: Integrating Different Courses and Web Material"
Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000
Trento, Italy, August 28-30, 2000
Lecture Notes in Computer Science 1892 Springer 2000
ISBN 3-540-67910-3

[Holmedahl-98] Holmedahl, V.; Smith, B. & Yang, T.
"Cooperative Caching of Dynamic Content on a Distributed Web Server"
7th International Symposium on High Performance Distributed Computing, 1998. Proceedings
28-31 July 1998 Page(s):243 – 250

[Hunter-01] Hunter, J. & Crawford, W.
"Java Servlet Programming – Second Edition"
O'Reilly, 2001
ISBN: 0-596-00040-5

[Ishida-98] Ishida, T.
"Towards Computation over Communities"
Community Computing and Support Systems, Social Interaction in Networked Communities [book based on the Kyoto Meeting on Social Interaction and Communityware, Kyoto, Japan, June 1998]
Lecture Notes in Computer Science 1519 Springer
ISBN 3-540-65475-5

[Jaczynski-99] Jaczynski, M. & Trousse, B.
"Broadway: A Case-Based System for Cooperative Information Browsing on the World-Wide-Web"
Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing
Lecture Notes in Computer Science 1624 Springer 1999
ISBN 3-540-66930-2

[Jaworsky-99] Jaworsky, J.
"Mastering JavaScript and JScript", SYBEX, 1999
ISBN: 0-7821-2492-5

[Joslin-00] Joslin, C.; Molet, T. & Magnenat-Thalmann, N.
"Advanced Real-Time Collaboration over the Internet"
Virtual Reality Software and Technology, VRST 2000
October 22-25, 2000, Seoul, Korea. ACM

[Koch-01] Koch, M.; Mancini, L.V. & Parisi-Presicce, F.
"On the Specification and Evolution of Access Control Policies"
6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)
May 3-4, 2001, Litton-TASC, Chantilly, Virginia, USA. ACM

[Li-00a] Li, S.F.; Spiteri, M.; Bates, J. et Al.
"Capturing and Indexing Computer-based Activities With Virtual Network Computing"
Proceedings of the 2000 ACM Symposium on Applied Computing
Villa Olmo, Via Cantoni 1, 22100 Como, Italy
March 19-21, 2000. ACM
ISBN 1-58113-239-5

[Li-00b] Li, S.F.; Stafford-Fraser, Q. & Hopper, A.
"Integrating Synchronous and Asynchronous Collaboration with Virtual Network Computing"
IEEE Internet Computing, Volume 4, Number 3, May/June 2000

[Lieberman-99] Lieberman, H.; Van Dyke, N.W. & Vivacqua, A.S.
"Let's Browse: A Collaborative Web Browsing Agent"
IUI'99, ACM
Redondo Beach, CA, USA, 1999

[Liechti-02] Liechti, O. & Sumi, Y.
"Editorial: Awareness and the WWW"
International Journal of Human Computer Studies, Volume 56, Number 1, January 2002

[Luotonen-98] Luotonen, A.
"Web Proxy Servers"
Prentice Hall PTR – Web Infrastructure Series, 1998
ISBN: 0-13-680612-0

[Maglajlic-99] Maglajlic, S.; Helic, D. & Scerbackov, N.
"A Practical Approach to Authoring Hypermedia Composites Used for Web Applications"
World Conference on the WWW and Internet
Honolulu, Hawaii, USA, October 24-30, 1999
Association for the Advancement of Computing in Eduction (AACE), Charlottesville, VA, USA, 1999
ISBN 1-880094-36-3

[Maglio-00] Maglio, P.P. & Farrell, S.
"LiveInfo: Adapting Web Experience by Customization and Annotation"
Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000
Trento, Italy, August 28-30, 2000
Lecture Notes in Computer Science 1892 Springer 2000
ISBN 3-540-67910-3

[Milner-84] Milner, R.
"Calculus for Communicating Systems"
Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA
July 9-11, 1984. Lecture Notes in Computer Science 197 Springer 1985
ISBN 3-540-15670-4

[Raymond-04] Raymond, D; Baudin, V; Kanenishi, K. et Al.
Distant e-learning using synchronous collaborative environment "Platine"
IEEE 6th International Symposium on Multimedia Software Engineering, MSE'2004
Miami (USA), December 13-15, 2004, pp.88-95

[Reif-01] Reif, G.; Kirda, E.; Gall, H. et Al.
"A Web-based peer-to-peer Architecture for Collaborative Nomadic Working"
10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises: WetIce'2001, Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises
MIT, Cambridge, Massachusetts, USA, 2001.

[Richardson-98] Richardson, T.; Stafford-Fraser, Q.; Wood, K.R. et Al
"Virtual Network Computing"
IEEE Internet Computing, Volume 2, Number 1, January/February 1998

[Rodriguez-02] Rodriguez-Peralta, L. M.; Villemur, T.; Drira, K. et Al
"Managing Dependencies in Dynamic Collaborations using Coordination Diagrams"
6th International Conference on Principles of Distributed Systems. OPODIS 2002
Reims, France, December 11-13, 2002
Studia Informatica Universalis 3 Suger, Saint-Denis, rue Catulienne, France
ISBN 2-912590-26-4

[Sakamoto-00] Sakamoto, R. & Kunifuji, S.
"Collaborative World Wide Web Browsing System through Supplement of Awareness"
4th International Conference on knowledge-Based Intelligent Engineering Systems & Allied Technologies, IEEE
Aug – Sep 2000, pp. 233 – 236

[Sanrach-00] Sanrach, C. & Grandbastien, M.
"ECSAIWeb: A Web-Based Authoring System to Create Adaptive Learning Systems"
Adaptive Hypermedia and Adaptive Web-Based Systems, International Conference, AH 2000
Trento, Italy, August 28-30, 2000
Lecture Notes in Computer Science 1892 Springer
ISBN 3-540-67910-3

[Sanranch-00] Sanranch, C. & Grandbastien, M.
"ECSAIWeb: A Web-Based Authoring System to Create Adaptive Learning Systems", AH'2000: LNCS 1892
Springer-Verlag Berlin Heideberg, 2000, pp. 214 – 226

[Schlichter-98] Schlichter, J.H.; Koch, M. & Xu, C.
"Awareness - The Common Link Between Groupware and Community Support Systems"
Community Computing and Support Systems, Social Interaction in Networked Communities [based on the Kyoto Meeting on Social Interaction and Communityware, Kyoto, Japan, in June 1998]
Lecture Notes in Computer Science 1519 Springer 1998
ISBN 3-540-65475-5

[Shen-92] Shen, H. & Dewan, P.
"Access Control for Collaborative Environments"
CSCW '92, Proceedings of the Conference on Computer Supported Cooperative Work
October 31 - November 4, 1992, Toronto, Canada. ACM

[Sidler-97] Sidler, G.; Scott, A. & Wolf, H.
"Collaborative Browsing in the World Wide Web"
8th Joint European Networking Conference
Edinburgh, May 12.-15. 1997
http://www.tik.ee.ethz.ch/~cobrow/papers/jenc8/jenc8.html

[Specht-00] Specht, M.
"ACE – Adaptive Courseware Environment"
Adaptive Hypermedia and Adaptive Web-Based Systems, International Conference, AH 2000
Trento, Italy, August 28-30, 2000
Lecture Notes in Computer Science 1892 Springer
ISBN 3-540-67910-3

[Steinfield-99] Steinfield, C.; Jang, C.J. & Pfaff, B.
"Supporting Virtual Team Collaboration: The TeamSCOPE System"
GROUP'99, ACM, Phoenix, Arizona, USA, 1999, pp. 81 – 90

[Takahashi-00] Takahashi, K. & Yana, E.
"A Hypermedia Environment for Global Collaboration"
IEEE Multimedia CSCW,
Oct. – Dec. 2000, pp. 36 – 47

[Tazi-01] Tazi S. & Evrard, F.
"Intentional structures of documents"
12th ACM Conference on Hypertext and Hypermedia
August 14-18, 2001, University of Aarhus, Århus, Denmark
ACM 2001

[Tolksdorf-01] Tolksdorf, R. & Glaubitz, D.
"XMLSpaces for Coordination in Web-based Systems"
10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises: WetIce'2001, Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises
MIT, Cambridge, Massachusetts, USA, 2001

[Twidale-95] Twidale, M.; Nichols, D.M.; Smith, G. et Al.
"Supporting Collaboration Learning during Information Searching"
Computer Supported Collaborative Learning (CSCL'95)
October 17-20, 1995
Indiana University, Bloomington, Indiana, USA.

[Twidale-97] Twidale, M.; Nichols, D.M. & Paice, C.D.
"Browsing is a Collaborative Process"
Information Processing and Management, Volume 33, Number 6, 1997
Elsevier

[Vasudevan-99] Vasudevan, V. & Palmer, M.
"On Web Annotations: Promises and Pitfalls of Current Web Infrastructure"
32nd Annual Hawaii International Conference on System Sciences (HICSS-32)
5-8 January, 1999, Maui, Hawaii
Track 2: Digital Documents
IEEE Computer Society, 1999
http://computer.org/proceedings/hicss/0001/00012/0001toc.htm

[Vogelsang-01] Vogelsang, L. & Carstensen, P.H.
"New Challenges for the Collaboration in Web-Based Information Systems Development"
10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001)
20-22 June 2001, Cambridge, MA, USA.
IEEE Computer Society 2001
ISBN 0-7695-1269-0

[Wang-99] Wang, W.
"Team-and-Role-Based Organizational Context and Access Control for Cooperative Hypermedia Environments"
Hypertext'99, ACM
Darmstadt, Germany, 1999

Références Web

[CADP] http://www.inrialpes.fr/vasy/cadp/

[CERN] http://public.web.cern.ch/Public/Welcome.html

[Firefox] http://www.firefox.com/

[HTML] http://www.w3.org/MarkUp/

[HTML401] http://www.w3.org/TR/1999/REC-html401-19991224

[HTML401] http://www.w3.org/TR/html4/

[HTTP] http://www.w3.org/Protocols/

[IANA] http://www.iana.org/

[Isabel] http://isabel.dit.upm.es

[ITU] http://www.itu.int/ITU-T/

[Java] http://java.sun.com/

[JSDT-20] http://java.sun.com/products/java-media/jsdt/index.jsp

[JSDT-21] https://jsdt.dev.java.net/

[K-Meleon] http://kmeleon.sourceforge.net/

[Lab@Future] http://www.labfuture.net/

[MOLM] http://main.placeware.com

[Mozilla] http://www.mozilla.org/

[MSIE] http://www.microsoft.com/

[NetDive] http://www.netdive.com

[Netscape] http://www.netscape.com/

[NSCA] http://www.ncsa.uiuc.edu/

[OMG] http://www.omg.org/

[Opera] http://www.opera.com/

[Platine] http://www.laas.fr/PLATINE/

[RFC0822] http://www.ietf.org/rfc/rfc0822.txt

[RFC1945] http://www.ietf.org/rfc/rfc1945.txt

[RFC2045] http://www.ietf.org/rfc/rfc2045.txt

[RFC2049] http://www.ietf.org/rfc/rfc2049.txt

[RFC2396] http://www.ietf.org/rfc/rfc2396.txt

[RFC2616] http://www.ietf.org/rfc/rfc2616.txt

[SDL] http://www.sdl-forum.org/

[Servlet] http://java.sun.com/products/servlet/

[SGML] http://xml.coverpages.org/sgml.html

[TCL] http://www.activestate.com/Products/ActiveTcl/

[Telelogic] http://www.telelogic.com/

[Tina] http://www.laas.fr/tina/

[Tomcat] http://jakarta.apache.org/tomcat/

[UML] http://www.uml.org/

[VNC] http://www.realvnc.com/

[VRVS] http://www.vrvs.org

[W3C] http://www.w3.org/

[WABX] http://linuxdemo.eisti.fr/aditri/index_fr.dim

[WebEx] http://www.webex.com

[WikiPedia] http://fr.wikipedia.org/wiki/Wiki

[WikiWikiWeb] http://c2.com/cgi/wiki?WikiWikiWeb

CoLab : Conception et Mise en Œuvre d'un Outil pour la Navigation Coopérative sur le Web

Cette thèse a pour objectif de concevoir un nouveau paradigme de navigation Web, permettant à un ensemble d'utilisateurs de naviguer de manière coopérative sur le Web. Ces utilisateurs, selon des règles de synchronisation simples mises à jour de manière dynamique, peuvent synchroniser l'affichage de pages Web dans leur navigateur.

Nous avons modélisé les différentes primitives de synchronisation proposées au moyen d'automates étendus et avons formalisé le modèle global de synchronisation par des réseaux de Petri. Une vérification formelle de ce modèle, grâce aux outils TINA et ALDEBARAN, a montré que les primitives de synchronisation rendaient effectivement le service attendu.

Partant d'une description informelle et intuitive de l'architecture du système de navigation coopérative, nous avons proposé une méthode pour la formaliser au moyen du profil UML/SDL supporté par l'outil TAU G2 de Telelogic. Nous avons détaillé l'architecture au moyen de différents diagrammes UML de ce profil et l'avons validé en comparant les traces de simulation obtenues avec les scénarios (diagrammes de séquences) élaborés au début de la phase de conception de notre système.

Nous avons développé une implémentation du système de navigation coopérative en Java, appelée CoLab, qui est à ce jour opérationnelle et qui supporte les principales primitives de synchronisation. Cette implémentation s'appuie sur la présence d'un serveur Proxy centralisé qui synchronise les utilisateurs selon les relations de synchronisation établies entre eux. Elle s'appuie également sur une technique originale de traduction des pages Web qui permet de rendre les actions de navigation détectables par le serveur Proxy. Nous avons réalisé des campagnes de mesure du temps moyen de rapatriement de ressources Web, en fonction du nombre d'utilisateurs enregistrés dans une session et de la taille des ressources, qui ont montré les bonnes performances de la plate-forme jusqu'à plus d'une centaine d'utilisateurs par session. S'appuyant sur ces résultats, nous avons proposé une solution pour répartir CoLab sur plusieurs serveurs Proxy. Finalement, nous avons montré comment rendre disponible le service de navigation coopérative sous la forme d'un service Web.

CoLab: Design and Implementation of a Tool for Browsing Collaboratively on the Web

This thesis' main goal is the design of a new Web browsing paradigm which allows a set of users to collaboratively browse the Web. These users can synchronize the display of Web pages in their browsers by the use of simple synchronization rules which are updated dynamically.

We have modelled all the different proposed synchronization primitives by using extended automata, and we have formalized the global synchronization model by using Petri nets. We have made a formal verification of this model by the use of the TINA and ALDEBARAN software tools, and we have shown that the synchronization primitives provide effectively the expected service.

Starting from an informal and intuitive description of the collaborative browsing system's architecture, we have proposed a method for formalizing it by using the UML/SDL profile supported by the Telelogic software tool Tau G2. We have detailed the architecture by using the different UML diagrams implemented in this profile, and we have validated them by comparing the obtained simulation traces with the scenarios (Sequence Diagrams) that we have proposed at the beginning of the design phase of our system.

We have developed an implementation of our collaborative browsing system using Java, called CoLab, which is actually operational and which supports the main synchronization primitives. This implementation is based on a centralized Proxy server which synchronizes the users according to the existing synchronization relations among them. It is also based on an original technique of translation of the retrieved Web pages which allows making the browsing actions of the users to be detectable by the Proxy server. We have carried out some performance tests campaigns in order to measure the average retrieval time of Web resources, according to the number of logged-in users in a session, and the size of the retrieved resources, which has shown a very good performance of the platform up to more than one hundred users per session. Based on these results we have proposed a solution to distribute CoLab among several Proxy servers. Finally we have shown how to make out collaborative browsing system available as a WebService.