MASTER EN SCIENCES INFORMATIQUES

Partage de ressources bioinformatiques hétérogènes: conception et implémentation d'une fédération de médiateurs

Dallons, Quentin; Buyle, Pierre

Award date:
2003

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année Académique 2002 - 2003

Partage de ressources bioinformatiques hétérogènes - Conception et implémentation d'une fédération de médiateurs

Pierre Buyle & Quentin Dallons

Mémoire présenté en vue de l'obtention du grade de Maître et Licencié en Informatique.

Résumé

Le domaine d'application de la bioinformatique est aussi varié qu'hétérogène. Pour le biologiste moléculaire, la bioinformatique constitue un outil de travail permettant d'effectuer des traitements élémentaires sur des données personnelles. Aujourd'hui, la situation de la bioinformatique se résume plutôt à une collection de sites offrant des services donnant accès à des sources de données biologiques.

Cependant, pour partager ces ressources et pour éviter la duplication systématique des sources de données, il devient impératif que les infrastructures offrant ces ressources communiquent de manière la plus efficace possible. Par ailleurs, récemment, beaucoup de solutions issues des TIC sont apparues et facilitent la distribution et la collaboration des systèmes d'informations (ex: OMG CORBA, Microsoft .NET, Sun J2EE, Services Web, etc.).

Laurent Debaisieux et Fernando Desouza, sous la direction de Vincent Englebert et de Marc Colet, ont conçu une fédération d'hôtes interconnectés permettant le partage de "Ressources Bioinformatiques Hétérogènes".

Dans ce mémoire, nous proposons un raffinement de l'architecture proposée par Laurent Debaisieux et Fernando Desouza en la rendant plus robuste et résistante à la montée en charge. Pour valider cette nouvelle architecture, nous proposons également un prototype.

Abstract

The application area of bioinformatics is diverse and heterogeneous. For the common molecular biologist, bioinformatics is a routine tool that allows basic tasks on personal biological data. The today situation of bioinformatics is merely a collection of individual sites providing a certain number of services and giving access to a certain number of data sources.

In order to share resources and to avoid systematic duplications of data sources we have to find a way to make computers communicate in an efficient way. Lots of IT solutions have appeared recently that facilitate the deployment of collaborating computers systems (e.g.: OMG's CORBA , Microsoft .Net, Sun J2EE, Web Services, ...).

Laurent Debaisieux and Fernando Desouza under the direction of Vincent Englebert and Marc Colet have drawn the first lines of an architecture and an implementation of a federation of interconnected hosts that allow "Heterogeneous Bioinformatics Resources" sharing.

The proposal of this master thesis is the refining of this architecture to make it more scalable and robust and providing a prototype to validate it.

Remerciements:

Nous désirons remercier le Professeur Vincent Englebert pour sa patience et sa disponibilité lors de l'élaboration de ce mémoire ainsi que la connaissance de qualité qu'il nous a transmise.
Un merci tout particulier à Marc Colet, Valérie Ledent, Daniel Van Belle et Richard Kamuzinzi qui nous ont accueillis chaleureusement au sein de l'unité de bioinformatique de l'Institut de Biologie et Médecine Moléculaire de l'Université Libre de Bruxelles. Un merci pour le temps souvent précieux qu'ils nous ont consacré pour répondre à nos nombreuses questions. Toute notre gratitude à Richard Kamuzinzi pour le service bioinformatique de test qu'il nous a implémenté.
Nous remercions également Yves Bontemps pour ses conseils quant à l'utilisation de LaTeX 2ε.
Merci à Aurélia, Jessica, nos parents et amis qui nous ont soutenus pendant les moments les plus difficiles.

Table des matières

Table des figures

Introduction

La bioinformatique est une discipline des *sciences de la vie* en pleine expansion. Sa thématique très large reflète la diversité du champ des recherches en biologie moléculaire. Cependant, elle ne se limite pas uniquement à cette dernière et révèle différents types d'activités tels que le traitement de données structurelles des macro-molécules, les comparaisons de séquences nucléiques ou protéiques, la conception de systèmes de persistance pour ces séquences ou pour l'archivage des collections de micro-organismes, le développement de nouvelles méthodes de modélisation, etc.

La situation actuelle de la bioinformatique du point de vue des apports en technologies de l'information et de la communication peut se résumer à une collection de sites individuels fournissant des services et des accès à des sources de données. Ces services sont très hétérogènes car ils sont constitués d'une part, de programmes "faits maison" écrits dans divers langages de programmation et reposant sur des formats de données très différents et peu spécifiés, et d'autre part, de programmes regroupés en packages commerciaux bien intégrés dont le coût des licences est problématique pour les institutions à but non lucratif.

La sécurité des données en bioinformatique pose aussi problème. En effet, peu de sites sont capables, aujourd'hui, de garantir la confidentialité des données soumises. Ceci constitue souvent l'argument justifiant que les industriels limitent leur collaboration avec d'autres institutions et maintiennent eux-mêmes leurs propres sources de données et leurs propres outils.

Récemment, beaucoup de solutions logicielles issues des technologies de l'information et de la communication sont apparues. Ces solutions facilitent le déploiement et la collaboration des systèmes d'information (par exemple: OMG CORBA, Microsoft .NET, Sun Java 2 Enterprise Edition, Services Web, etc.). Leur utilisation dans le cadre de la problématique de la bioinformatique assurerait une augmentation de la sécurité, une réduction de la réplication systématique des outils et des sources de données mais aussi une amélioration des conditions d'utilisation de ces outils pour le biologiste. En effet, pour celui-ci, la bioinformatique constitue généralement un outil parmi tant d'autres lui permettant d'effectuer différentes tâches sur les données *biologiques* issues de ces travaux.

Dans cet objectif d'amélioration de l'ergonomie des outils et d'optimisation des ressources, Marc Colet (Département de Biologie Moléculaire de l'Université Libre de Bruxelles) et Vincent Englebert (Institut d'informatique des Facultés Universitaires Notre-Dame de la Paix de Namur) ont tracé ensemble les premières lignes du Projet FEDERGEN: "Permettre l'utilisation des systèmes existants, sans aucune modification, tout en fournissant un ensemble de mécanismes efficaces de distribution et d'accès aux services bioinformatiques."

Pour atteindre ce but, un premier travail, réalisé par Laurent Debaisieux et Fernando Desouza dans le cadre de leur mémoire de licence en informatique à horaire décalé, a traité du partage des ressources bioinformatiques hétérogènes [DD03]. Dans ce mémoire, ils ont effectué une première analyse des exigences du futur système et proposé une architecture et son prototype permettant la création d'une fédération de partage de ressources bioinformatiques.

C'est dans la continuité de ce mémoire, que nous avons mené notre stage de fin d'études à l'Institut de Biologie et Médecine Moléculaire de l'ULB. Notre réflexion s'est portée sur l'amélioration de l'architecture de fédération proposée par Laurent Debaisieux et Fernando Desouza ainsi que sur l'implémentation d'un prototype fonctionnel.

Ce mémoire a pour objectif de présenter de manière critique notre démarche de conception et d'implémentation de cette fédération de partage de ressources bioinformatiques hétérogènes.

Nous avons structuré ce travail en deux parties. La première partie présente un état de l'art de la bioinformatique, de l'informatique distribuée et du Projet FEDERGEN. Ainsi, dans un premier chapitre, nous présenterons la bioinformatique et l'état de sa normalisation. Dans un second chapitre, nous aborderons les avantages justifiant la distribution des systèmes d'information ainsi que les techniques permettant d'y parvenir. Pour terminer, nous traiterons du premier travail d'analyse, effectué par Laurent Debaisieux et Fernando Desouza, fournissant une première approche du partage de ressources bioinformatiques hétérogènes.

La seconde partie, quant à elle, couvre la réalisation de notre travail. Nous présenterons dans un premier chapitre notre méthodologie d'analyse. Ensuite, un second chapitre traitera de l'analyse des exigences réalisée lors de notre stage à l'IBMM. Celle-ci sera suivie, dans le troisième chapitre, de la conception logique de notre architecture et, dans un quatrième et un cinquième chapitre, de sa conception physique et de son implémentation en un prototype. Le dernier chapitre proposera, quant à lui, des critiques portant sur notre architecture et des propositions d'améliorations envisageables dans le cadre d'une future évolution du projet.

Première partie

Etat de l'art

Chapitre 1

La Bioinformatique

1.1 Introduction

Cette section constitue la synthèse des références suivantes: [CAA99, Cou02, GJ01a, Tis01].

La bioinformatique est née de la convergence de la biologie et de l'informatique. Les recherches menées en biologie, et plus particulièrement en génomique, ont apporté, dès le milieu du 20ème siècle une masse d'informations difficile à gérer manuellement. L'informatique, en tant que science du traitement de l'information, apporte tout naturellement les outils utiles à la gestion de ces données.

Les apports de l'informatique à la discipline sont multiples. Au début, ce sont les techniques de stockage des données, puis les bases de données relationnelles qui ont intéressé les biologistes à la recherche d'outils nouveaux pour la gestion de leurs données. De nos jours, les bioinformaticiens mettent au point de nouvelles méthodes d'analyses "biologiques" se basant aussi bien sur l'informatique pratique que sur des bases plus théoriques telles que la théorie des graphes ou les statistiques avancées. De même, de nombreux outils destinés directement aux biologistes utilisent largement les résultats des recherches traitant de la problématique de la visualisation d'importantes quantités de données au moyen d'interfaces homme-machine.

Plus qu'un ensemble de techniques informatiques au service des biologistes, la bioinformatique est la science de l'analyse des informations utiles pour la compréhension de la biologie. C'est une façon de faire de la biologie, une approche de la discipline sous l'angle de la modélisation et de la simulation.

Pour illustrer notre propos, nous pouvons reprendre la définition suivante: *Bioinformatics is conceptualising biology in terms of molecules (in the sense of physical chemistry) and applying "informatics techniques" (derived from disciplines such as applied maths, computer science and statistics) to*

understand and organise the information associated with these molecules, on a large scale. In short, bioinformatics is a management information system for molecular biology and has many practical applications. [LGG01]

Les informations analysées et traitées en bioinformatique sont essentiellement issues de la génomique et de la biologie moléculaire. On y retrouve donc des séquences d'*ADN* brutes, des séquences de protéines, des modèles de structures macro-moléculaires, des séquences de génomes et toutes sortes d'autres données liées aux génomes. Si ces données représentent aujourd'hui la majorité de l'information en bioinformatique, des données autres sont aussi à prendre en compte. Tout résultat ou produit d'expériences ou d'observations biologiques peut constituer une source d'informations qui est, ou pourra être un jour, largement stockée, diffusée ou traitée.

Tout comme ces données, la plupart des applications de la bioinformatique sont elles aussi issues de la génomique. L'analyse de l'ADN humain devrait, par exemple, permettre de mettre au point de nouveaux médicaments et des traitements pour certaines maladies. Toujours dans le domaine médical, l'étude des similarités de structures entre molécules permet également de concevoir de nouveaux médicaments beaucoup plus simplement et rapidement qu'auparavant.

De grands progrès ont pu être réalisés grâce à la bioinformatique au niveau de la classification des espèces vivantes. Les propriétés génétiques des espèces sont mieux connues et mieux classifiées, ce qui permet une meilleure et plus simple utilisation.

Avec le développement de la discipline, de plus en plus de domaines des *sciences de la vie* voient, dans la bioinformatique, de nouvelles possibilités. Petit à petit, la bioinformatique entre dans des laboratoires de recherches de tout type apportant tantôt de simples supports pratiques, tantôt de nouvelles méthodes et modèles de travail.

La bioinformatique est une discipline en pleine croissance. Et, de ce fait, de plus en plus d'applications différentes seront nécessaires et de plus en plus de ressources devront leur être attribuées. Alors que ces ressources sont actuellement fortement centralisées, l'augmentation des besoins va probablement entraîner une plus grande distribution. En plus de la conception de nouveaux outils liée à l'ouverture de la discipline, les développements futurs en bioinformatique doivent donc permettre de nouveaux modes de distribution capables de supporter l'augmentation des besoins en ressources des scientifiques.

1.2 Ressources en bioinformatique

On peut distinguer deux grandes catégories de ressources en bioinformatique: d'une part, les ressources de stockage d'informations, typiquement les énormes bases de données contenant un ou plusieurs génomes, d'autre part,

les outils de traitement de l'information. L'utilisation de ces deux types de ressources est généralement liée, les informations traitées par les outils sont souvent directement issues des bases de données, et ce parfois, en de très gros volumes.

Les principales sources de données pour les bioinformaticiens sont les importantes bases de données des principaux centres et laboratoires de recherche telles que celles proposées par le National Center for Biotechnology Information (NCBI)[fBI03]. Pour leur stockage des données, chaque centre ou laboratoire utilise sa propre solution. Les échanges massifs de données, eux, se font via l'échange de fichiers de grande taille respectant divers formats de structure de l'information (voir section 1.3).

En plus de l'échange en gros volume, l'accès aux informations stockées dans ces bases de données peut se faire de manière ponctuelle. Leurs utilisateurs ont ainsi la possibilité de consulter ces bases par l'extraction, à l'aide d'outils, de données répondant à des critères de recherche. Une deuxième possibilité d'accès à ces bases est l'utilisation d'interfaces web pour des outils de traitement de données accédant directement aux bases de données. L'outil d'extraction (voir exemples à la section 1.5.1) se voit ainsi proposé ses données.

Pour conserver les données de recherche ou de traitement, on utilise des fichiers respectant les mêmes formats que les fichiers utilisés pour l'échange en gros volume des données mais de taille plus réduite. Différents modes d'accès à ces fichiers peuvent être utilisés. En plus de l'utilisation directe d'outils de traitement, ces fichiers peuvent parfois être partagés entre plusieurs utilisateurs via des modes d'accès similaires à ceux proposés par les fournisseurs de larges banques de données. Par ailleurs, la gestion de la sécurité de ces systèmes n'étant jamais étudiée dans une optique de collaboration entre organisations, elle entraîne une réplication quasi systématique des outils de traitement et des sources de données en vue d'en assurer, en interne, la sécurité.

Les outils de traitement utilisés en bioinformatique se présentent généralement sous la forme de programmes en ligne de commande avec de nombreux paramètres d'exécution (voir exemples à la section 1.5.2). Leur volume pouvant se révéler important, les entrées et sorties de données se font généralement via des fichiers séparés du programme plutôt que par interaction avec l'utilisateur. Pour simplifier l'utilisation de ces outils, des interfaces web sont souvent utilisées. Ces interfaces ne sont généralement pas très évoluées et offrent plutôt un accès graphique simplifié aux multiples paramètres des outils qu'elles supportent. Des programmes présentant des interfaces graphiques existent également mais, tout comme les interfaces web, ils se contentent généralement d'offrir un mode d'accès graphique aux paramètres des outils en ligne de commande.

1.3 Le problème de la Normalisation

Le travail des chercheurs qui utilisent la bioinformatique repose souvent sur l'échange et l'exploitation d'informations récoltées par d'autres ou issues d'analyses plus ou moins complexes. L'exploitation de ces données s'opère au moyen de divers outils. Cependant, pour que ces outils puissent traiter une information, celle-ci doit être structurée de façon à ce que le programme puisse la comprendre, et donc selon un format qu'il connaît. De même, lorsqu'un bioinformaticien souhaite écrire un nouvel outil de traitement, il doit décider dans quel format seront présentées les données d'entrée et de sortie. Pour garantir un maximum d'échanges, il est idéal, dans un maximum de cas, d'utiliser le même formalisme pour le même type de données.

Malheureusement, la formalisation des données et le développement des outils en bioinformatique ainsi que la maturation des principales disciplines utilisatrices de celle-ci ont eu lieu en même temps. Les différentes découvertes scientifiques apportant de nouveaux types d'informations à stocker et à traiter sont apparues avant même que les théories qui y sont liées ne soient complètement finalisées. Les différents formats de fichiers utilisés ont donc évolué au gré des découvertes et des besoins, de façon anarchique, en réponse aux besoins techniques de l'instant.

Il en résulte de nombreux formats de fichiers différents, répondant à des besoins différents et supportés par des outils différents. Il n'existe par contre que peu de tentatives de formalisation, au sens large, de l'information en bioinformatique. Les formats spécifiés ne permettent que le stockage et le traitement d'informations fort spécifiques. De même, les différents fournisseurs d'outils ne se sont que tardivement intéressés au partage d'informations entre les outils issus de champs de recherche différents.

En pratique, chaque outil, ou suite d'outils s'accompagne de son propre format de fichiers. Les différents programmes d'une suite d'outils utilisent le même format, mais les programmes de deux suites différentes ne peuvent pas communiquer directement entre eux (voir exemples à la section 1.5.3). Pour échanger de l'information entre ces programmes, les utilisateurs doivent avoir recours à des outils de conversion de formats plus ou moins complexes. Pour traiter ces données, l'utilisateur a donc besoin de connaître de nombreux détails techniques concernant ces outils et leurs formats de fichiers. Ceci nuit bien évidemment à l'interopérabilité de ces outils, au partage de données en bioinformatique mais aussi à leur ergonomie.

De même, lorsqu'ils désirent faire collaborer différents programmes, les bioinformaticiens ne disposent pas d'une formalisation des données suffisante et doivent dès lors composer avec les différents formats existants. Les normes existantes répondent souvent à des besoins techniques et elles ne peuvent pas toujours être adaptées à l'utilisation qu'un développeur souhaiterait en faire. Ceci entraîne une plus grande complexité de création de nouveaux outils.

Comme nous l'avons déjà laissé sous-entendre dans l'introduction de ce

mémoire, la bioinformatique va devoir passer d'un mode de fonctionnement centralisé à un mode de fonctionnement basé sur des systèmes d'information distribués. L'utilisation de tels systèmes va forcer la collaboration entre des outils conçus différemment et parfois même gérés indépendamment les uns des autres. Il sera donc impératif d'assurer un maximum d'interopérabilité entre les outils nouvellement créés et existants. De plus, la conception d'applications distribuées robustes nécessitera, entre autre, une bonne spécification des données. Dès à présent, il paraît, évident que la définition d'une norme pour les données en bioinformatique, est une nécessité.

Le travail est bien sûr gigantesque mais, même si elles sont peu utilisées, il existe déjà des normes pour les données bioinformatiques. La plupart de ces normes ont avant tout un objectif pratique : fournir aux bioinformaticiens des outils communs pour faciliter le développement d'applications. Des projets tels que BioPerl [Bio03c], BioJava [Bio03b] et BioPython [Bio03d] proposent des ensembles d'outils de traitements bioinformatiques réutilisables pour leur langage de programmation respectif. Le fait que la création d'outils réutilisables nécessite une certaine formalisation des données nous amène à considérer ces projets comme des tentatives de définitions de normes. Mais il s'agit plus de normes techniques pour le traitement de données que de normes d'échanges et de structuration de l'information. De plus, la spécification des données n'est pas leur but premier.

Il existe également des normes d'échange ou d'interaction entre applications telles que BioCorba [Bio03a] [1], BSML [BSM03] ou BSA [Groa]. Mais, à nouveau, ce sont souvent des normes techniques, conçues pour résoudre uniquement le problème technique de l'échange des données. La norme BSA se différencie cependant des autres normes. En effet, la normalisation du format des données et leur définition font partie de ses objectifs. Elle n'a cependant pas été conçue dans le cadre d'un projet d'application réelle, mais dans le but de rencontrer les besoins actuels et futurs d'un maximum d'applications bioinformatiques.

1.4 La norme BSA

1.4.1 Présentation

La norme **BSA** (Biomolecular Sequence Analysis)[Groa] est une spécification établie par l'**OMG** [2] pour le domaine des *Sciences de la Vie*. La version actuelle (1.0) a été publiée en juin 2001 par le groupe *Life Science Research Task Force* rassemblant de nombreux acteurs du domaine.

1. Le site de ce projet était accessible en décembre 2002, depuis il semble que le nom de domaine ait été repris par une entreprise sans relation avec le projet ou la bioinformatique.

2. L'Object Management Group (OMG) [Grob] est un consortium d'organismes privés et publiques qui produit des spécifications visant à faciliter les développements d'architectures logicielles exploitant les avantages du paradigme orienté objet.

La norme BSA se décompose en deux modules:

Le module `DsLSRBioObjects` définit les interfaces d'un modèle de représentation de données de biologie moléculaire.

Le module `DsLSRAnalysis` précise les interfaces pour les outils d'analyse ainsi que leur mode d'interaction.

Sans rentrer dans les détails, nous pouvons dire que le module `DsLSRBioObjects` de la norme BSA décrit les interfaces d'objets d'encapsulation des données pour les domaines de la biologie moléculaire. Il fournit, par exemple, des interfaces d'objets pour les séquences nucléiques et protéiques mais également pour les diverses informations attachées à ces séquences. Au final, le module spécifie tous les objets pour les données que pourrait avoir à manipuler un bioinformaticien.

De son côté, le module `DsLSRAnalysis` nous intéresse plus directement puisqu'il définit comment seront produits et utilisés les objets de données spécifiés. Ce module spécifie les interfaces nécessaires aux applications d'analyses de séquences en utilisant un modèle d'analyse générique. En plus des outils eux-mêmes, le module fournit les moyens nécessaires pour retrouver les paramètres d'entrées, les résultats ou les propriétés d'un outil précis. Les principales interfaces spécifiées sont données dans la figure 1.1.

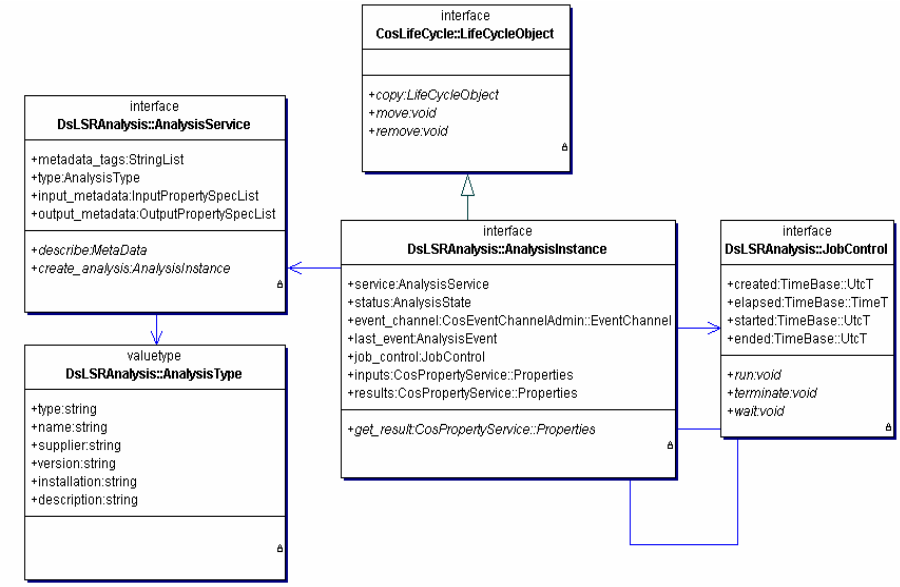


FIG. 1.1 – *Diagramme des interfaces principales du module* `DsLSRAnalysis`

Un `AnalysisService` est une représentation logique d'un outil particulier d'analyse BSA disponible. Les attributs d'un `AnalysisService`, permettant l'identification de l'analyse qu'il peut opérer, sont donnés par son `AnalysisType` et la liste de ses paramètres d'entrées et résultats par ses `In-`

`putPropertySpecList` et `OutputPropertySpecList`. Lorsqu'un client souhaite utiliser un service, il s'adresse à l'`AnalysisService` correspondant qui lui fournit alors un `AnalysysInstance`. L'`AnalysysInstance` représente une et une seule exécution d'un outil d'analyse, c'est l'objet responsable de la réalisation effective de l'analyse et de la gestion des ses résultats. Le contrôle de l'exécution de l'analyse elle-même se fait via l'élément `JobControl` de l'`Analysisinstance`.

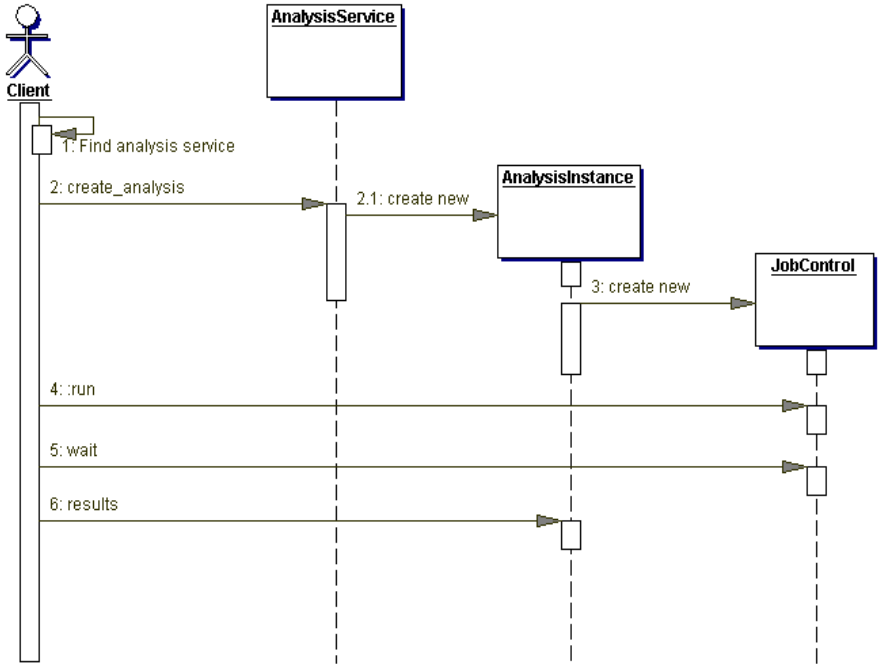


FIG. 1.2 – *Diagramme de séquence des interactions pour l'utilisation d'un* `AnalysisService`

La norme BSA spécifie aussi l'interaction entre ses différentes interfaces au moyen d'un diagramme de séquence. Ainsi, la figure 1.2 présente les interactions entre les divers objets et le client lors de l'invocation d'un service de façon synchrone[3], par exemple. Le client demande d'abord à l'`AnalysisService` correspondant à l'analyse qu'il souhaite effectuer de créer un `AnalysisInstance`. Il demande ensuite au `JobControl` de cet `AnalysisInstance` de démarrer l'analyse elle-même. Une fois l'analyse achevée, le client récupère le résultat en le demandant à l'`AnalysisInstance`.

3. D'autres modes d'invocation du service peuvent être envisagés mais ils reposent sur d'autres normes de l'OMG que nous n'avons pas abordées: les `EventChannel` pour des appels asynchrones

Les interfaces de la norme BSA sont spécifiées dans le langage de description d'interfaces IDL[4] de l'OMG et reposent, en partie, sur d'autres spécifications de l'OMG (`CosLifeCycle`, `CosPropertyService`, etc.) fortement liées à CORBA[5]. L'utilisation de ces spécifications vise, avant tout, la généricité par rapport à la solution d'implémentation choisie.

1.4.2 Implémentation et statut de la norme

Malgré ses qualités, la norme BSA n'a pas rencontré l'approbation de la communauté des bioinformaticiens. Elle se veut indépendante de tout choix d'implémentation mais repose sur d'autres spécifications de l'OMG fort proches de CORBA. La complexité de certains mécanismes de généricité la rend trop lourde par rapport à la plupart des besoins actuels en bioinformatique. Elle est donc peu pratique pour la réalisation rapide d'applications, ce qui décourage son emploi.

La norme est également arrivée trop tôt, puisque, lors de sa publication, les implémentations CORBA utilisées en bioinformatique[6] ne fournissaient que partiellement les fonctionnalités requises par la norme BSA, rendant son utilisation plus qu'hasardeuse. En définitive, même si le module `DsLSRAnalysis` a été utilisé par plusieurs projets[7] ainsi que, probablement, par des entreprises de services bioinformatiques, le module `DsLSRBioObjects` de la norme n'a quasi jamais été utilisé.

Il reste que, même si nous devons nous montrer prudents quant à son utilisation, la norme BSA est à peu près la seule partagée par plusieurs projets et suffisamment libre de tout détail technique. Le module `DsLSRAnalysis` nous semble convenir pour la gestion des interactions entre clients et fournisseurs de services, tout en restant suffisamment indépendant du module `DsLSRBioObjects` pour permettre une architecture ouverte à d'autres normes ou standards de représentation de données.

1.5 Exemples d'applications

1.5.1 Extraction de données

Les figures 1.3 et 1.4 illustrent l'extraction de séquences nucléiques relatives au gène humain ADAM2 (fertiline-beta humaine) dans les banques

4. L'Interface Description Language (IDL) est un langage défini par l'OMG. Ce langage permet de définir des interfaces de modules de manière indépendante de tout langage de programmation.

5. Common Object Request Broker Architecture (CORBA) est une norme qui a été proposée par l'OMG et qui définit une architecture permettant l'intégration et la collaboration d'applications écrites dans des langages et opérant dans des environnements (hardware ou software) très différents (voir section 2.3).

6. Principalement des implémentations libres pour Perl ou Java.

7. Dont le projet AppLab de l'EBI [Ins01]

EMBL et Genbank au moyen du même outil d'interrogation SRS de BEN mais en version ligne de commande (figure 1.3) ou interface Web (figure 1.4).

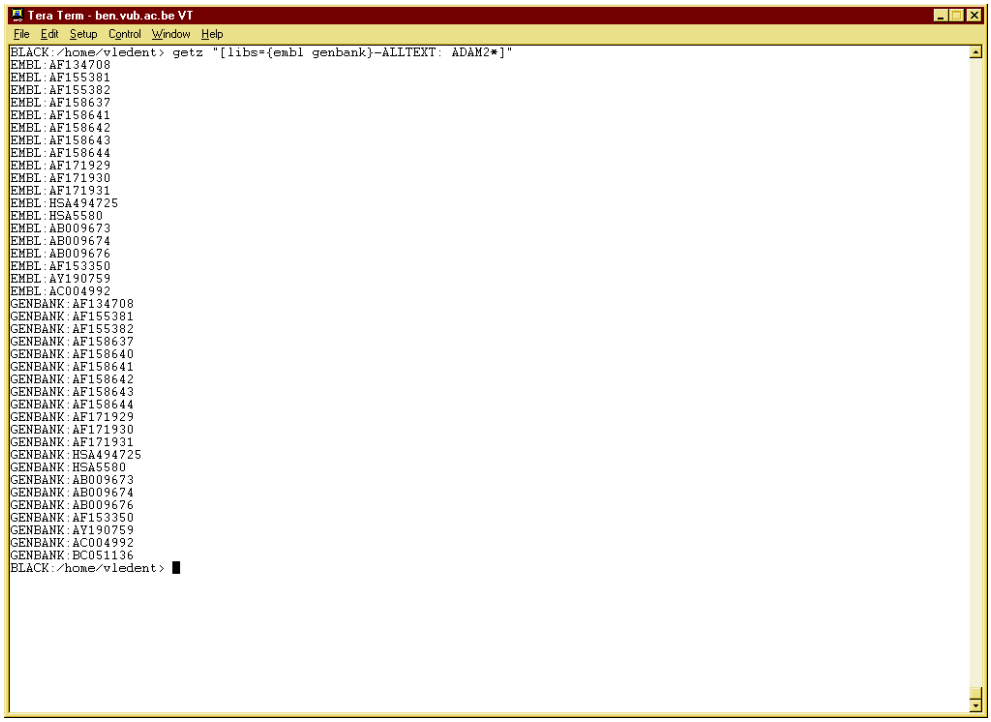


FIG. 1.3 – *Extraction de séquences nucléiques relatives au gène humain ADAM2 (fertiline-beta humaine) dans les banques EMBL et Genbank au moyen de l'outil d'interrogation SRS de BEN (version shell)*

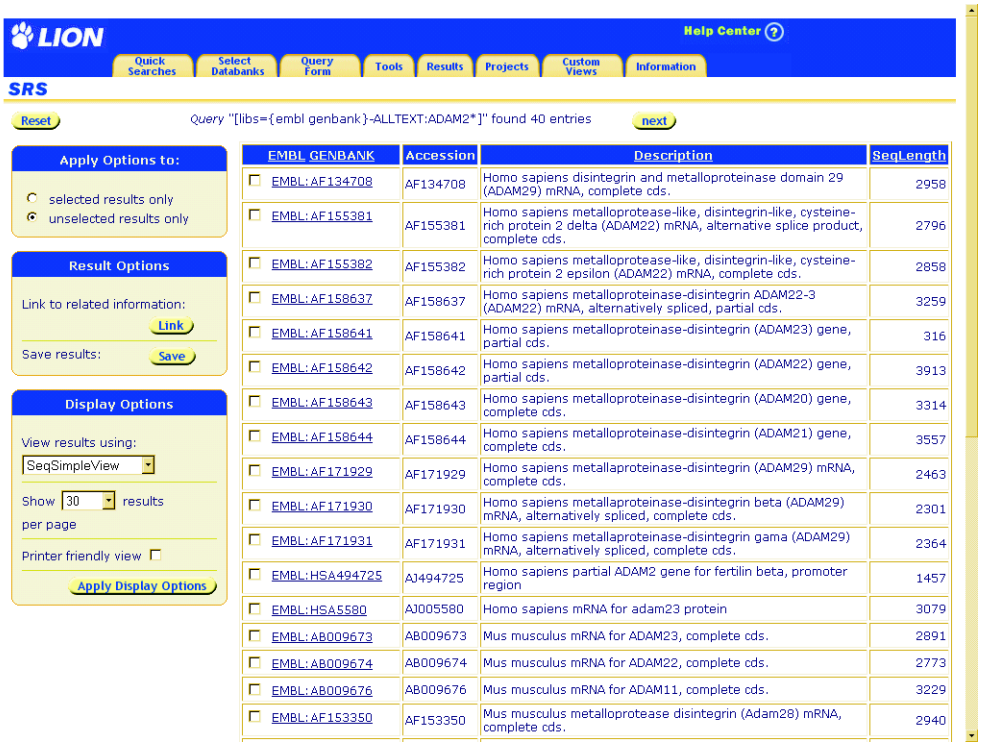


FIG. 1.4 – *Extraction de séquences nucléiques relatives au gène humain ADAM2 (fertiline-beta humaine) dans les banques EMBL et Genbank au moyen de l'outil d'interrogation SRS de BEN (version interface web)*

1.5.2 Analyse de données

Les figures 1.5, 1.6, 1.7 et 1.8 illustrent la saisie des paramètres et la réception des résultats pour un meilleur alignement local[8] entre les séquences embl: X56677 (ARNmessager du gène humain MyoD) et embl: M84918 (ARNmessager du gène MyoD de Mus musculus (souris)) en utilisant le programme matcher de EMBOSS en version shell (figure 1.5 et figure 1.6) ou en version interface Web (figure 1.7 et figure 1.8).

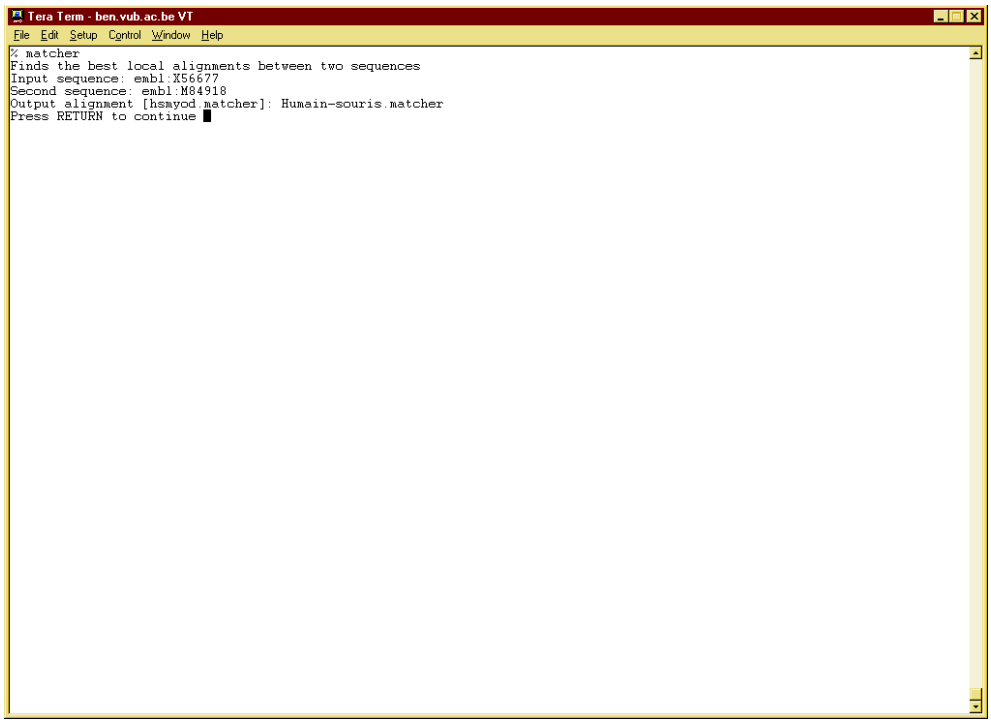


FIG. 1.5 – *Saisie des paramètres pour un meilleur alignement local en utilisant le programme matcher de EMBOSS (version shell) entre les séquences embl: X56677 (ARNmessager du gène humain MyoD) et embl: M84918 (ARNmessager du gène MyoD de Mus musculus (souris)).*

8. Alignement : technique superposant des séquences différentes à des fins de comparaison. [GJ01b]

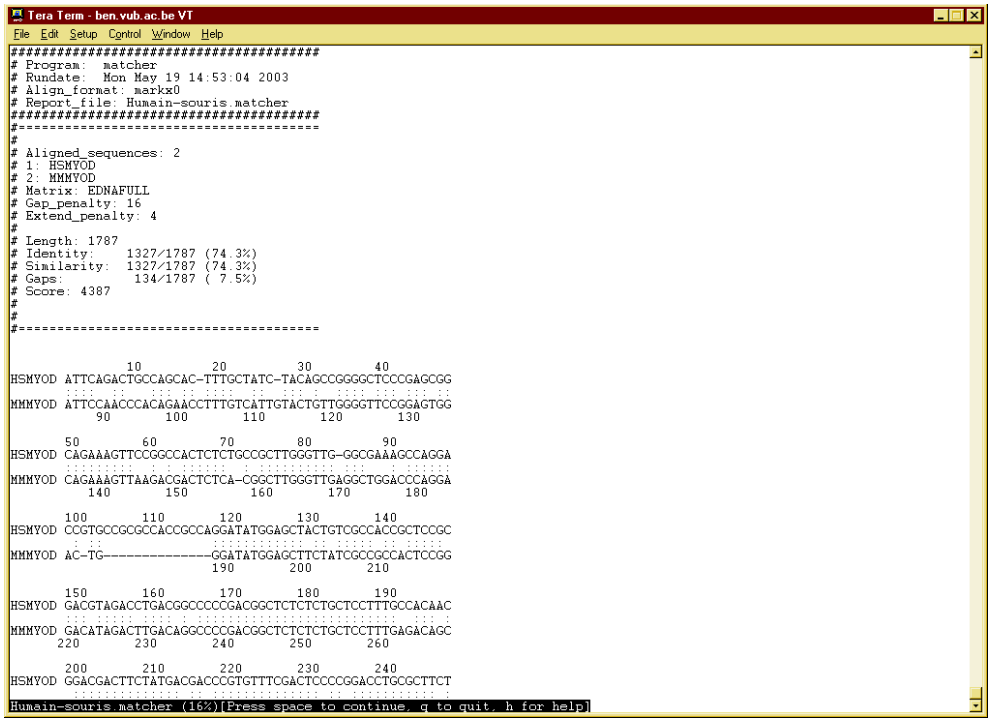


FIG. 1.6 – *Résultats pour un meilleur alignement local en utilisant le programme matcher de EMBOSS (version shell) entre les séquences embl: X56677 (ARNmessager du gène humain MyoD) et embl: M84918 (ARNmessager du gène MyoD de Mus musculus (souris)).*

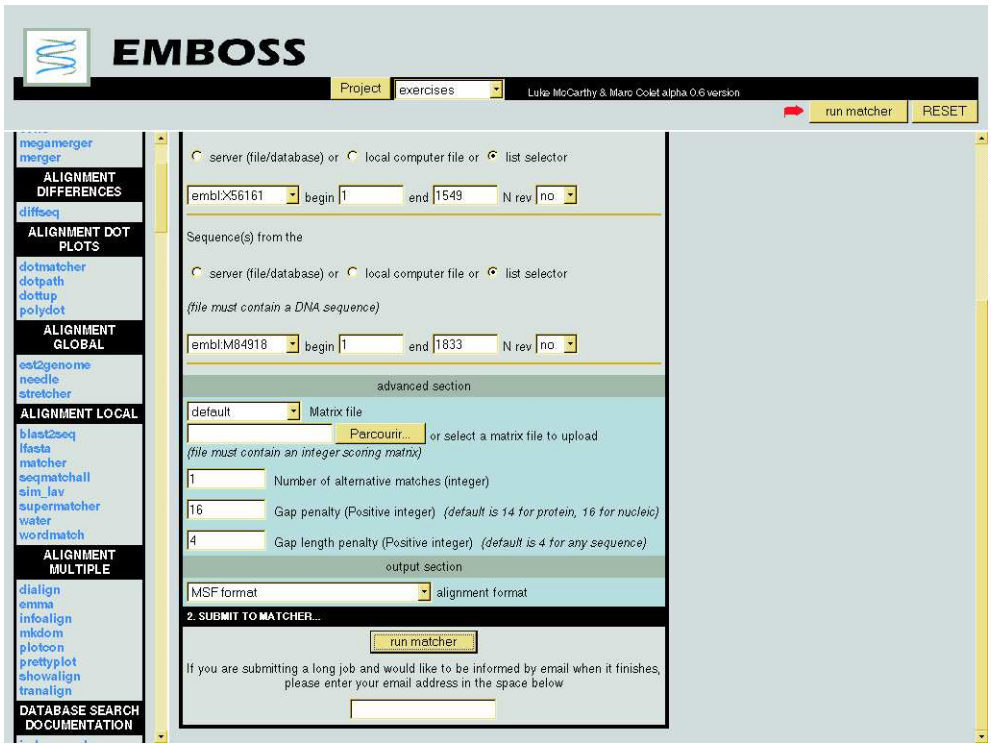


FIG. 1.7 – *Saisie des paramètres pour un meilleur alignement local en utilisant le programme matcher de EMBOSS (version interface Web) entre les séquences embl: X56677 (ARNmessager du gène humain MyoD) et embl: M84918 (ARNmessager du gène MyoD de Mus musculus (souris)).*

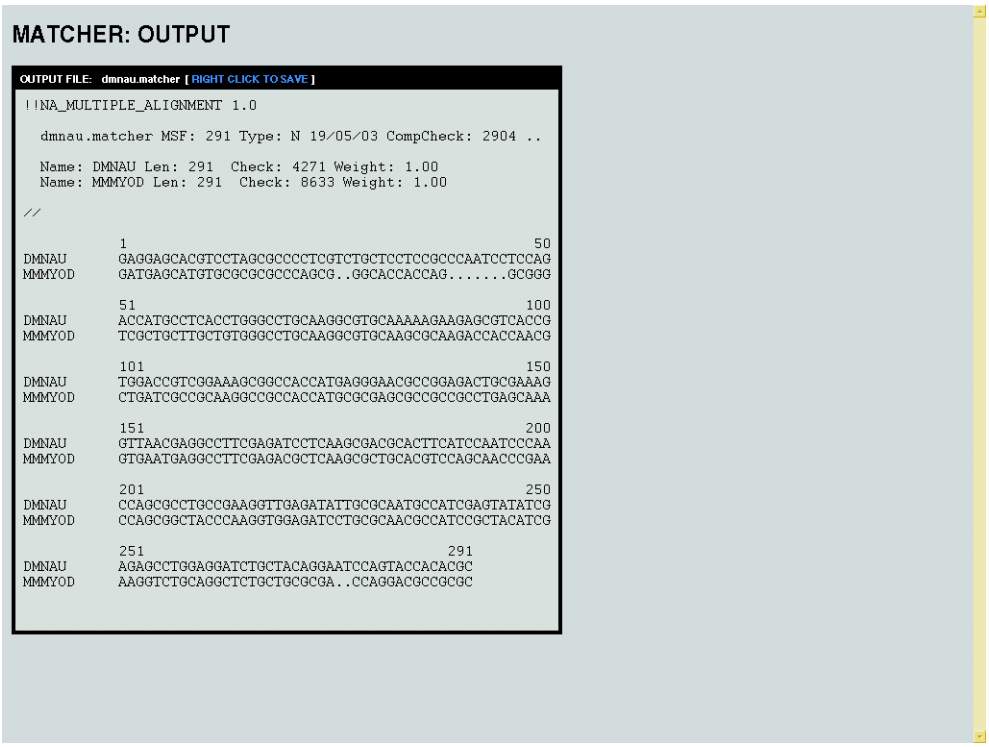


FIG. 1.8 – *Résultats pour un meilleur alignement local en utilisant le programme matcher de EMBOSS (version interface Web) entre les séquences embl: X56677 (ARNmessager du gène humain MyoD) et embl: M84918 (ARNmessager du gène MyoD de Mus musculus (souris)).*

1.5.3 Conversion de formats

L'exemple suivant illustre le problème de format de données incompatibles.

Par exemple, le programme d'alignement multiple Clustal souvent utilisé préalablement à une analyse phylogénétique[9] fournit ses résultats au format illustré sur la figure 1.9, tandis que le programme d'analyse phylogénétique Phylip demande les mêmes informations au format d'entrée illustré sur la figure 1.10.

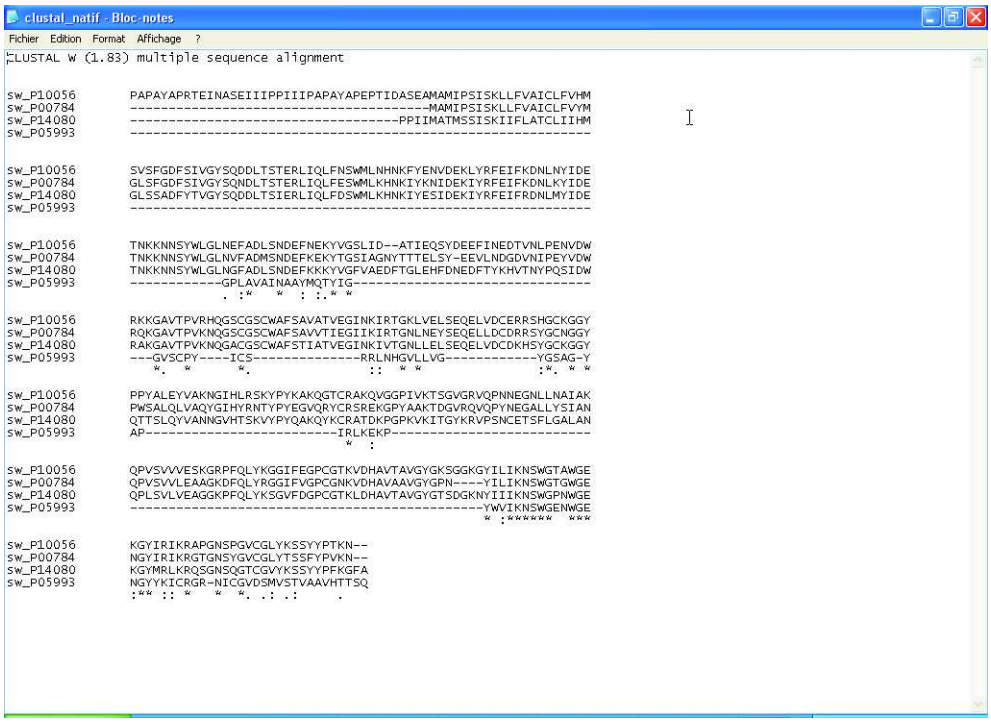


FIG. 1.9 – *Format de résultat natif de l'application d'alignement multiple Clustal*

Heureusement, le programme Clustal est capable de fournir sur demande le format Phylip en résultat. Cependant, dans d'autres cas, il est parfois nécessaire de passer par une adaptation faite "à la main" des formats de données ou par l'emploi d'un script (Perl bien souvent) écrit préalablement.

9. C'est l'inférence, à l'aide de modèles théoriques, de la filiation entre espèces, sur base de la comparaison de leur diversité moléculaire.[GJ01b]

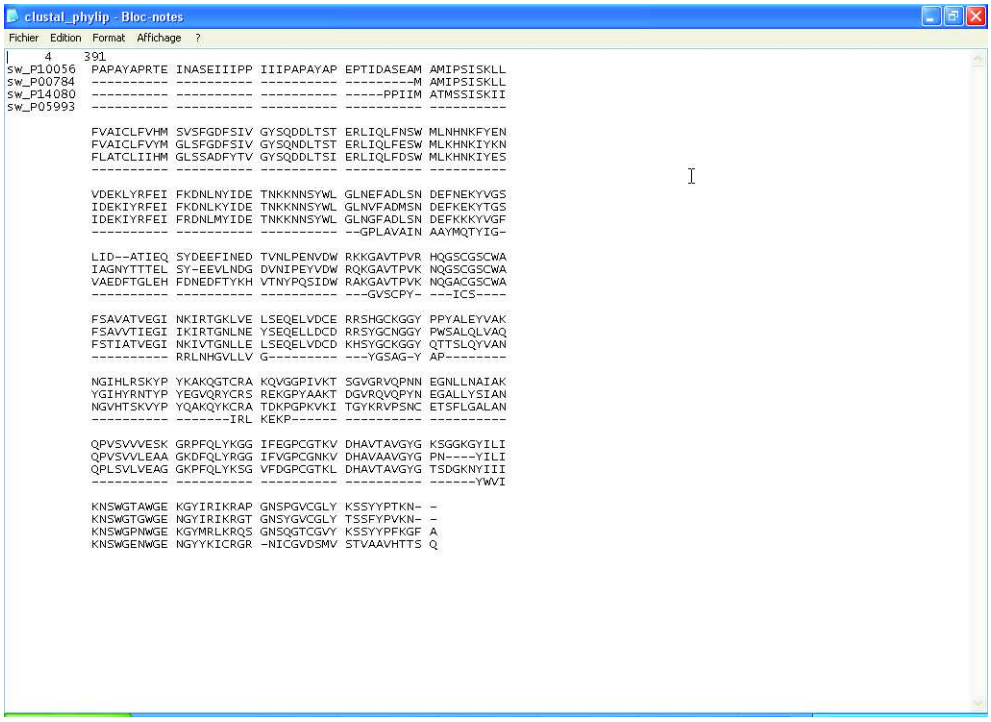


FIG. 1.10 – *Format d'entrée de l'application d'analyse phylogénétique Phylip*

1.6 Conclusion

La bioinformatique regroupe au sens large les *Sciences de la vie* et l'informatique en une nouvelle discipline. La bioinformatique ne consiste pas seulement en l'utilisation de l'outil "informatique" par les scientifiques, mais aussi en un nouveau mode de recherche: une approche de la biologie sous l'angle de la modélisation et de la simulation. Si son principal champ d'application réside dans la génomique, elle n'y est pas restreinte; tout autre champ des *Sciences de la vie* peut également s'y rattacher.

La bioinformatique est une discipline en pleine évolution et ses besoins sont croissants. En effet, de plus en plus de données et de types de données différents sont traités par un nombre d'outils de plus en plus important. Cette augmentation de la taille des données et l'augmentation du nombre d'outils permettant de les exploiter entraînent leur décentralisation, leur collaboration et leur formalisation. Leur décentralisation et leur collaboration sont nécessaires pour assurer leur croissance actuelle et future en terme de ressources; leur formalisation, quant à elle, est nécessaire pour l'échange d'informations et l'interopérabilité entre les différents outils. Pour atteindre ces buts, la norme BSA a fourni un premier formalisme des données et des interactions entre outils issus de la bioinformatique; formalisme sur lequel peuvent être construites des applications distribuées plus complexes et plus riches en terme de fonctionnalités.

Chapitre 2

L'informatique distribuée

2.1 Introduction

Ce chapitre a pour objectif de présenter les avantages liés à la distribution des systèmes d'informations et les différentes technologies disponibles pour y parvenir. Ces technologies sont les différents *middlewares* tels que CORBA, Sun Java 2 Enterprise Edition, Microsoft .NET et Services Web ainsi que les systèmes Peer-To-Peer et GRID.

2.2 La distribution des systèmes d'information

Cette section est basée sur le cours de "Conception des systèmes d'information coopératifs" [Eng02].

Nous allons tout d'abord définir les notions de "système distribué" et de "composant". Ensuite, nous proposerons un ensemble d'exigences auxquelles un système d'information distribué doit répondre pour apporter un réel gain par rapport à un système d'information centralisé.

Qu'est-ce qu'un système distribué?
"Un système distribué est une collection d'hôtes autonomes qui sont connectés par l'intermédiaire d'un réseau. Chaque hôte héberge et exécute un ou plusieurs composants tout en supportant un middleware, qui permet aux composants du système de coordonner leurs activités d'une telle façon que les utilisateurs perçoivent le système comme une capacité de traitement unique et intégrée. On oppose généralement système distribué à système centralisé." [Eng02]

Cette définition est claire mais le terme "composant" est encore flou. En parcourant la littérature spécialisée, il est possible de définir un composant comme suit:

"A component in an architecture is a unit of computation or a data store. Components may be as small as a single procedure or as large as an entire application. Each component may require its own data or execution space, or it may share them with other components." [MNTN00]

Cette définition n'est pas forcément des plus claires. Pour la simplifier, nous dirons qu'un composant est une unité élémentaire (dont la granularité est variable) de traitement, de stockage de données ou les deux, au sein du système d'information. Les composants peuvent être en relation entre eux. Par relation, on entend la communication ou le partage de l'espace d'exécution entre composants.

Maintenant que les termes "systèmes distribués" et "composants" sont définis, il est pertinent de se poser la question suivante: "Qu'apportent les systèmes d'information distribués par rapport aux systèmes d'information centralisés plus classiques?".

Pour trouver une réponse à cette question, il suffit d'examiner les exigences auxquelles un système d'information distribué est capable de répondre.

Ces exigences sont les suivantes:

- Une plus grande souplesse lors de l'évolution: un système d'information doit être capable de faire face à la montée en charge de l'ensemble du système pour répondre à des délais de traitements imposés par les utilisateurs. Le système distribué doit être ouvert à l'évolution de ses composants suite à l'apparition de nouveaux langages de programmation ou de modifications de l'architecture initiale du système.
- Une plus grande facilité pour accéder et partager des ressources hétérogènes: un système distribué permet l'utilisation conjointe de composants d'origines différentes tels que des "legacy components"[1], des composants développés entièrement en interne ou des composants provenant de tiers. Ces ressources hétérogènes (Matériel, Logiciel ou Données) doivent être rentabilisées en permettant à plusieurs personnes d'y accéder simultanément ainsi qu'en concurrence. Pour y parvenir, le système distribué doit aussi garantir la pertinence,l'intégrité et la confidentialité des données . Le système doit également assurer la sécurité des composants et gendarmer les accès concurrents.
- Une fiabilité accrue: un système distribué est plus sensible aux pannes qu'un système centralisé; cela est dû à sa complexité en terme des technologies mises en oeuvre. D'un autre côté, il est plus fiable car il peut muter en fonction du contexte. En effet, il est possible d'activer un composant sur un autre hôte que celui d'origine, ce dernier étant

1. Anciens composants existants et devant être réutilisés

soit en panne soit saturé. La réplication des données sur des hôtes distincts quant à elle permet d'en assurer la sécurité.

- Une meilleure réutilisation de ce qui a été développé précédemment: un composant de par sa conception est aisément utilisable dans d'autres contextes ce qui permet de rentabiliser au mieux les frais liés à sa conception.
- De meilleures performances globales: diviser la charge de travail induite par une tâche et la répartir sur plusieurs unités de traitement permet d'en diminuer le temps de traitement total.

Pour parvenir à distribuer un système d'information, celui-ci doit reposer sur un ou des *middleware(s)*. Un *middleware* consiste en une couche d'abstraction de haut niveau garantissant tant la transparence à l'utilisateur dans son utilisation du système qu'au développeur dans son utilisation et sa conception du système.

Ainsi, cette transparence devra répondre aux critères suivants:

- La transparence des accès: l'accès à un composant (local ou distant) doit être possible tout en ignorant les détails de son implémentation.
- La transparence de la localisation: l'accès à un composant doit être possible sans se préoccuper de sa localisation physique (hôte, adresse IP, etc.).
- La transparence de la migration: une modification de la distribution des composants sur les différents hôtes du système n'influence en rien son utilisation.
- La transparence de la réplication: la présence de composants dupliqués pour des raisons de précaution, d'amélioration de la disponibilité (qualité de service) et de tolérance aux pannes ne doit en rien perturber l'utilisation du système.
- La transparence de l'évolution: l'ajout de composants, d'hôtes ou autres ne doit en rien perturber le fonctionnement général du système.
- La transparence de la concurrence: les utilisateurs du système ne doivent pas percevoir que l'exécution de certains composants nécessite des accès concurrents à certaines ressources.
- La transparence de la performance: les mécanismes (réplication, migration, load-balancing, etc.) mis en oeuvre pour accroître les performances du système distribué doivent rester invisibles pour l'utilisateur.
- La transparence des échecs: le système doit cacher au moyen de mécanismes (réplication, migration, transaction, etc.) l'échec éventuel de certains composants mais aussi garantir les propriétés de *liveness* [2] et *safeness* [3] du système.

2. La demande de l'utilisateur se produira tôt ou tard.
3. La demande de l'utilisateur se produira correctement ou pas du tout.

– La transparence de la relocalisation: le système doit être capable de supporter le déplacement d'un composant, alors qu'il est actif, en cours d'exécution. Ce déplacement sera invisible pour l'utilisateur.
– La transparence de la persistance: l'utilisateur ne doit pas avoir conscience de l'état persistant ou éphémère d'un composant.

Les systèmes d'information distribués constituent une solution évolutive, performante et adaptée à un grand nombre de situations rencontrées aujourd'hui dans la conception des systèmes d'informations. Malgré une complexité supérieure, ils assurent une plus grande fiabilité tout en conservant la transparence d'utilisation à l'utilisateur et au développeur du système.

2.3 Common Object Request Broker Architecture

2.3.1 Object Management Group

Créé en 1989, l'Object Management Group (OMG) [Grob] est un consortium de plus de 800 membres constitué essentiellement d'organismes privés (entreprises de software, de hardware, etc.) et publics (universités, instituts de recherche publics, etc.). L'OMG produit des spécifications telles que l'Unified Modeling Langage (UML) et le Common Object Request Broker Architecture (CORBA) afin de faciliter les développements d'architectures logicielles, éventuellement distribuées, tout en exploitant les avantages du paradigme orienté objet.

2.3.2 La norme CORBA

Common Object Request Broker Architecture (CORBA) est une norme qui a été proposée par l'OMG et qui définit une architecture permettant l'intégration et la collaboration d'applications écrites dans des langages et opérant dans des environnements (hardware ou software) très différents. Cette architecture consiste en un *middleware* orienté objet qui rend invisible au développeur la gestion de bas niveau (système d'exploitation, matériel, communication réseau, etc.). Les spécifications de la norme CORBA en sont actuellement à la version 3.0.2. La norme se compose de l'*Object Request Broker* (ORB), des *Objects Services*, des *Common Facilities* et des *CORBA Domains*.

Il est important de remarquer que l'OMG ne définit que des spécifications pour le bus CORBA et les différents services qu'elle propose. Elle ne se soucie pas des détails d'implémentation. Il est dès lors possible d'implémenter au moyen de CORBA des solutions tout à fait spécifiques à des problèmes particuliers.

L'Object Request Broker

L'ORB est un bus de communication qui permet aux développeurs d'échanger des objets de façon transparente tout en garantissant une interopérabilité maximale entre diverses plateformes matérielles et logicielles. L'ORB assure également une compatibilité entre un grand nombre de langages de programmation existants.

Les Objects Services

Les Objects Services CORBA consistent en un ensemble de services à valeur ajoutée mis à disposition au-dessus de l'ORB. On peut y trouver [Dan02]:

- **Un service d'annuaire** qui offre la possibilité de trouver des objets par des noms symboliques.
- **Un service de courtier** qui offre la possibilité de trouver des objets en fonction de leurs caractéristiques.
- **Un service d'événements** qui permet à des objets producteurs d'événements de les envoyer à des objets consommateurs d'événements.
- **Un service de notification** qui offre le mécanisme de publication et souscription aux producteurs et consommateurs.
- **Un service de persistance des objets** qui permet un stockage persistant des objets.
- **Un service de gestion du cycle de vie des objets** qui offre des outils de gestion tels que la création, la copie, le déplacement et la destruction d'objets.
- **Un service de gestion de la concurrence** qui offre les outils tels que les verrous pour la gestion de la concurrence entre objets.
- **Un service d'externalisation** qui offre un mécanisme standard pour fixer ou extraire des objets du bus.
- **Un service de relation** qui offre la gestion de relations (appartenance, inclusion, emploi, etc.) que l'on peut créer dynamiquement entre objets CORBA et permettant de modéliser des liens entre objets.
- **Un service de transaction** qui offre un mécanisme de transaction entre objets respectant les propriétés "ACID"[4].
- **Un service de propriétés** qui permet d'associer dynamiquement des valeurs nommées à des objets.

4. ACID pour Atomique, Consistante, Isolée et Durable. Une transaction est Atomique si elle s'effectue complètement ou pas du tout, Consistante si elle préserve l'intégrité des données, Isolée si elle ne se soucie pas de l'existence d'autres transactions et Durable si son effet est permanent. [Eng02]

- **Un service de requêtes** qui offre la possibilité d'interroger les attributs d'objets.
- **Un service de temps** qui offre un temps universel sur le bus.
- **Un service de sécurité** qui offre l'identification et l'authentification des clients, le chiffrement des communications et le contrôle d'accès aux objets.
- **Un service de collection** qui offre des structures de manipulation d'objets sous forme de collection (listes, piles, tas, etc.) ainsi que leurs itérateurs.
- **Un service de version** qui offre la possibilité de gérer les différentes versions d'objets sur le bus.
- **Un service de messagerie** qui offre un modèle de communication proche du "Message Oriented Middleware"[5].
- **Un service de licences** qui permet de facturer l'utilisation des objets.

Les Common Facilities

C'est un ensemble d'applications qui résolvent des problèmes communs à de nombreuses architectures distribuées telles l'administration de système, la gestion des utilisateurs, la gestion des tâches, etc. [Eng02]

Les CORBA Domains

Ce sont des spécifications de composants métiers s'appliquant à des domaines particuliers tels que la Biologie moléculaire(BSA), le contrôle du trafic aérien, le *streaming* Audio/Video, les infrastructures à clés publiques (PKI), la gestion de *Workflow*, des services de recherches bibliographiques, la gestion de différentes devises monétaires, etc. [Eng02, Grob].

Le CORBA Component Model

Au 'Workshop on component-oriented programming at ECOOP 1996', Clemens Szyperski et Cuno Pfister ont défini un composant logiciel comme suit: "Un composant logiciel est une unité de composition dotée d'interfaces spécifiées et dont les relations de dépendances avec le contexte sont rendues explicites. Un composant logiciel peut être déployé indépendamment et être sujet à une composition par une tierce entité." [SP96]

Dans [Dan02], Jérôme Daniel fournit la définition suivante: "En résumé, on caractérise un composant comme étant un élément pouvant servir à la

5. Un Message Oriented Middleware (MOM) est un middleware particulier qui offre un système d'échange de messages fiable basé sur des queues. Une de celles-ci permet d'enregistrer des requêtes afin de les mettre à la disposition du serveur lorsque celui-ci est disposé à les traiter. Une autre queue permet, par la suite, au client de récupérer les résultats lorsqu'ils deviennent disponibles. [Eng02]

base à la conception d'une application. Les objets forment les composants et les composants forment les applications. Un composant atomique n'est pas un simple objet car les composants sont également personnalisables à l'aide de propriétés et caractérisés par des fonctionnalités additionnelles comme le déploiement ou la composition."

Ainsi, un composant CORBA est constitué des éléments suivants permettant aux clients et autres composants d'interagir avec lui [Eng03, Dan02]:

- Les facettes sont les interfaces exportées par un composant.
- Les réceptacles sont les points d'entrée nommés qui permettent d'interfacer le composant avec des objets typés.
- Les sources d'événements émettent des événements d'un type donné vers des consommateurs ou vers des canaux d'événements.
- Les puits d'événements sont les points d'entrée dans lesquels peuvent être déposés des événements d'un type donné.
- Les attributs sont des valeurs nommées qui permettent de paramètrer le composant dynamiquement.

2.3.3 L'architecture CORBA

L'architecture CORBA côté client

L'architecture CORBA côté client (figure 2.1) se compose des éléments suivants:

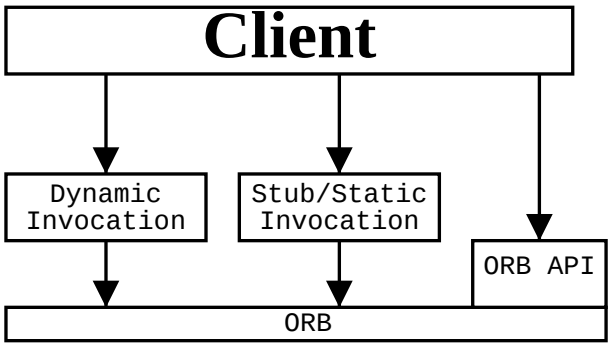


FIG. 2.1 – *Schéma de l'architecture CORBA côté client [Eng02]*

- ORB: L'ORB permet le transport des invocations entre les différents objets (se trouvant éventuellement sur des hôtes différents) au moyen du protocole *Internet Inter-ORB Protocol* (IIOP). Seules quelques

fonctions élémentaires sont accessibles par les applications "utilisateur".

– Le Stub/Static Invocation Interface (SII): Le *Stub* est le *Proxy* permettant à l'application "utilisateur" d'accéder de manière transparente à l'objet distant. Ce *Stub* est le résultat de la projection de la description des interfaces en IDL[6] dans le langage d'implémentation. Le *Stub* aura pour mission d'emballer les requêtes, de déballer les résultats et de réaliser les appels nécessaires à l'ORB pour traiter la requête. Un même objet distant peut avoir plusieurs *Stubs*, un par langage de projection (C++, Java, Cobol, etc.).

– Le Dynamic Invocation Interface (DII): ce mécanisme plus complexe permet de construire des requêtes dynamiquement, c'est-à-dire que les applications clientes découvrent les interfaces des composants distribués qui les entourent au moment de leur exécution.

L'architecture CORBA côté serveur

L'architecture CORBA côté serveur (figure 2.2) se compose des éléments suivants:

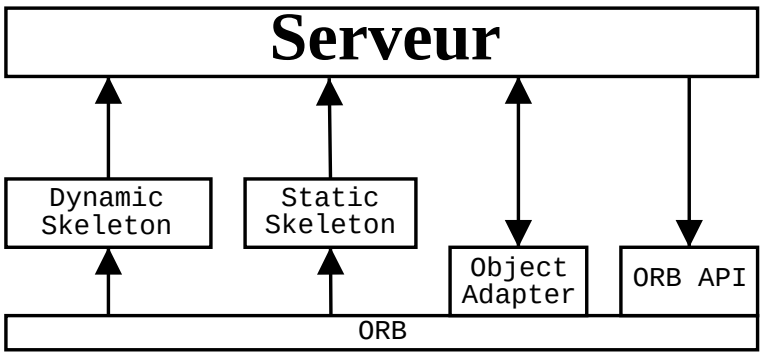


FIG. 2.2 – *Schéma de l'architecture CORBA côté serveur [Eng02]*

– Le Static Skeleton Interface: ce composant a pour mission de déballer les requêtes des clients pour les fournir à l'objet d'implémentation distant mais aussi d'emballer les résultats. Ce composant est généré

6. L'Interface Description Language (IDL) est un langage défini par l'OMG. Ce langage permet de définir des interfaces de composants de manière indépendante du langage. Cependant, il est possible de générer automatiquement les interfaces dans un langage orienté objet particulier (voir tab. 2.1).

sur base de la description en IDL des interfaces et est facultatif pour chaque objet distant.

– Le Dynamic Skeleton Interface: ce composant répond aux requêtes des clients. Il peut répondre aux requêtes sans que les *Skeletons* statiques ne soient disponibles. C'est le pendant du DII client mais côté serveur.

– L'Object Adapter: ce composant doit gérer les objets distants du serveur et les rendre accessibles (activation des objets) au client lors d'un appel. Il est aussi chargé de les désactiver.

Services essentiels de l'ORB

Les services essentiels de l'ORB sont:

– L'Interface Repository : ce service fournit des objets persistants offrant la description des interfaces et modules définis dans le système. L'Interface Repository contient une version de chaque interface. Il est permis à des composants de même type de cohabiter même si leurs interfaces sont de versions différentes.

– L'Implementation Repository: Ce service est spécifique à chaque fournisseur d'ORB et, permet entre autre, de stocker les informations sur les classes offertes et les instances d'objets.

– Les protocoles *General Inter-ORB Protocol* et *Internet Inter-ORB Protocol*: le *General Inter-ORB Protocol* (GIOP) est un protocole réseau à usage général, il est indépendant de la couche transport sous-jacente (TCP/IP ou autres). Il est simple à implémenter, générique et efficace à l'utilisation. GIOP définit la représentation physique *Common Data Representation* (CDR) des types IDL ainsi que la représentation des *Interoperable Object Reference* (IOR) [7]. Enfin, il spécifie un ensemble de pré-requis sur la couche transport: cette dernière doit être orientée connexion, fiable, *streamable* (dans le sens où la couche transport doit permettre de considérer la couche comme une suite continue de bytes); la couche transport doit pouvoir également avertir la couche applicative d'une déconnexion. Pour finir, GIOP impose un mode opératoire de la couche transport calqué sur TCP/IP.

  L'*Internet Inter-ORB Protocol* (IIOP) est la mise en oeuvre du protocole GIOP au-dessus de TCP/IP [Eng02] .

7. L'IOR est la représentation standardisée des références d'objets distants dans la norme CORBA.

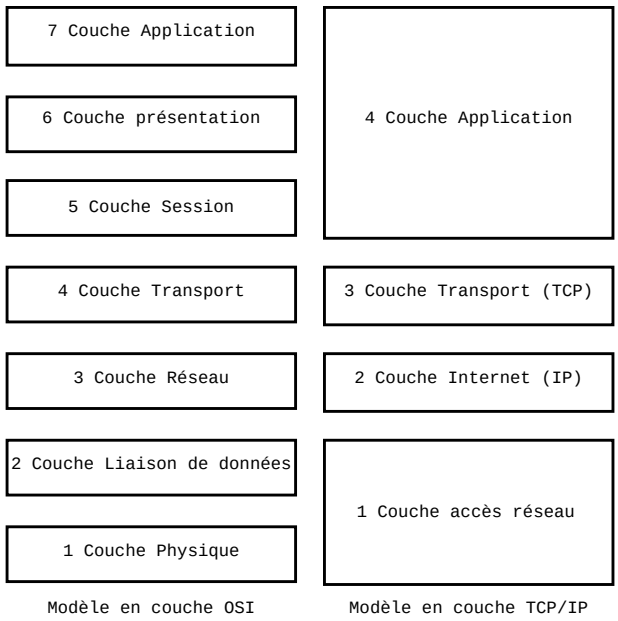


FIG. 2.3 – *La communication réseau selon les modèles OSI et TCP/IP*

Remarque: Le protocole TCP/IP ne respecte pas le modèle de communication en couche OSI (figure 2.3). TCP/IP se limite à 4 couches : la couche accès réseau (Physique + Liaison de Donnée), la couche Internet (IP), la couche transport (TCP) et la couche application (Session + Présentation + Application). [Tan02]

2.3.4 CORBA en pratique

Dans la pratique, le développeur décrit en IDL, les interfaces des objets qu'il désire distribuer. Ensuite, il passe ses interfaces IDL dans un compilateur cible qui va projeter ces interfaces IDL dans le langage d'implémentation (Java, C++, Cobol, etc.). Lors de cette projection, les amorces clientes (*Stub*) et mes amorces serveurs (*Skeleton*) seront aussi générées et lui permettront de faire des appels distants sans se soucier des couches inférieures
.
L'OMG a normalisé les projections (*mappings*) du langage de description d'interface (IDL) vers différents langages d'implémentation (voir tableau 2.1).

2.3.5 Critiques

Bien que CORBA présente les avantages suivants:

– La possibilité de collaboration entre diverses plateformes logicielles ou

TAB. 2.1 – *Exemple de* mapping *IDL vers C++ et Java [Eng02]*

CORBA/IDL	**C++**	**Java**
module	namespace	package
type élémentaire	définition de type	type élémentaire
constante	constante	attribut constant
définition de type	typedef	classe
énumération	enum	classe
structure	structure	classe
union	classe	classe
tableau	tableau	tableau
séquence	classe	tableau
interface	classe	interface
référence d'objet	instance	instance
attribut	fonctions d'accès	fonctions d'accès
opération	fonction	fonction
invocation	appel	appel
passage de paramètre	val/&/	valeur/Holder
exception	classe	classe
short	CORBA::Short	short
unsigned short	CORBA::Ushort	short
long	CORBA::Long	int
unsigned long	CORBA::Ulong	int
long long	CORBA::LongLong	long
unsigend long long	CORBA::ULongLong	long
float	CORBA::Float	float
double	CORBA::Double	double
long double	CORBA::LongDouble	pas encore défini
boolean	CORBA::Boolean	boolean
octet	CORBA::Octet	byte
char	CORBA::Char	char

matérielles constitue un des énormes avantages de CORBA. CORBA s'insère dans les différents langages de programmation de manière transparente et respectueuse de ceux-ci. CORBA s'utilise exactement de la même façon que les librairies ou classes du langage.

– CORBA est une norme libre d'implémentation, les développeurs sont libres et peuvent l'utiliser dans des situations spécifiques. Ils ne sont également pas obligés d'implémenter toutes les spécifications. L'architecture et les services de CORBA sont modulaires et peuvent s'ajouter au fur et à mesure des besoins rencontrés.
– CORBA est très complet, il suffit d'examiner tout ce qui est prévu par les *CORBA services*, les common facilities et les *CORBA domains* pour s'en rendre compte.
– CORBA offre de bonnes performances. En effet, il permet assez facilement la mise en oeuvre d'outils de load-balancing et de réplication d'objets sur le bus.
– L'utilisation du langage de description d'interface IDL est un plus pour

une plus grande indépendance vis-à-vis des langages et plateformes.
- Les implémentations des différents ORB respectant la norme CORBA sont interopérables indépendamment des plateformes logicielles et matérielles choisies.

Il présente également des inconvénients, tels que:
- CORBA est peut-être trop complet et peut rebuter les utilisateurs.
- A cause de sa complétude, il peut être complexe à déployer.
- Il est commercialement peu à la mode.
- Il souffre d'un énorme défaut vis-à-vis des *firewalls*. En effet, les appels CORBA se font en ouvrant des connexions multiples sur des ports TCP/IP différents. Cette spécificité peut poser des problèmes dans la gestion des ports d'un éventuel *firewall*. Des solutions existent pour résoudre ce problème mais elles sont souvent spécifiques à un fabriquant d'ORB.

2.4 Sun Java 2 Enterprise Edition

Cette section constitue la synthèse de références suivantes: [jGu, Micb, Eng03].

"Sun Java 2 Enterprise Edition" (J2EE) est un framework quatre tiers (Présentation - Services - Business Objects - Persistance) basé sur les "Enterprise Java Beans" (EJB) de Sun Microsystems. Les EJB de Sun offrent un environnement d'exécution distribué, sécurisé, transactionnel et persistant pour l'exécution de composants Java .

2.4.1 L'architecture EJB

EJB est une spécification propriétaire de Sun. Les environnements d'exécution J2EE/EJB sont implémentés par Sun mais peuvent être implémentés également par d'autres entreprises (IBM, WebLogic, Borland, Exolab, etc.).

Ces *Beans* Java peuvent être de deux types:
- Java Bean : Le *Java Bean* est un composant Java qui doit respecter un certain nombre de règles d'écriture (persistance des propriétés, génération d'événements et traitement d'événements) [Micc, Lok].
- Enterprise Java Bean : Un *Enterprise Java Bean* est un composant dans une architecture distribuée, il s'exécute dans un environnement adapté et sécurisé en utilisant les outils EJB tels que RMI, JNDI, JDBC et JTS.

L'architecture EJB reprend les éléments Java suivants:

- Le service *Remote Methode Invocation/Internet Inter-ORB Protocol* (RMI/IIOP) permet l'invocation de méthodes distantes en Java et utilise le protocole IIOP (voir section 2.3.3) défini par l'OMG. L'architecture EJB offre ainsi une compatibilité avec le protocole de CORBA et aussi une interopérabilité `EJB-CORBA`.
- Le service *Java Naming and Directory Interface* (JNDI) permet l'utilisation d'un annuaire de noms pour trouver un composant EJB.
- Le service *Java DataBase Connectivity* (JDBC) assure la gestion des connexions à un service de persistance des données.
- Le service *Java Transaction Service* (JTS) et l'api *Java Transaction Api* (JTA) assurent la gestion des transactions respectant les propriétés ACID entre composants Java distribués.
- Le service *Java Messaging Service* (JMS) est un service de messagerie transactionnel, asynchrone, tolérant aux pannes et évolutif.
- Les *Java Servlets* et *Java Server Pages* (JSP) permettent de créer des extensions aux serveurs http pour exécuter des traitements côté serveur.
- Le service *Java Mail* offre la possibilité d'envoyer des mail indépendamment de la plateforme.
- Les *Connectors* sont des "boîtes noires" fournies par un constructeur permettant l'introduction de composants non EJB dans l'architecture.
- L'*Extensible Markup Language* (XML)[8] décrit les propriétés des composants EJB.
- L'interopérabilité avec des composants non Java est possible.
- Les *Entity Beans* représentent des objets qui possèdent un état persistant. Ce sont des objets concrets du domaine d'application. Le système a la responsabilité de les préserver au moyen des transactions, de la gestion de la persistance, etc.
- Les *Session Beans* représentent une activité initiée par une application cliente. Cette session consomme des *Entity Beans* et fait l'objet d'une transaction mais le système ne tente pas de la préserver. Les sessions peuvent être avec ou sans état. Cet état, s'il existe, n'est pas préservé.

2.4.2 Critiques

L'architecture EJB offre des outils similaires à ceux spécifiés depuis longtemps par la norme CORBA. Les services tels que ceux d'annuaire (EJB JNDI et le service d'annuaire CORBA), de transactions (EJB JTS et le service de transaction CORBA), la persistance des objets (*Entity Beans* et le

8. Extensible Markup Language (XML). XML est un format texte dérivé de SGML. Il a été conçu à l'origine pour la publication à grande échelle. Cependant il joue un rôle croissant dans l'échange de données diverses sur le Web et ailleurs [Cona].

service de persistance des objets CORBA) sont autant d'exemples présents dans les deux technologies dont la paternité revient à CORBA.

Malgré le fait que les architectures à composants tel que EJB ont le défaut d'être lourdes en terme de ressources [Eng03], EJB a l'avantage de faciliter la gestion des composants (sécurité, déploiement, etc.) tout en garantissant une interopérabilité avec CORBA.

2.5 Microsoft .NET et COM+

Cette section constitue la synthèse de références suivantes: [Jab, Mica].

.NET est un framework quatre tiers (Présentation - Services - Business Objects - Persistance) basé sur la technologie COM+ de Microsoft. Les Objets COM+ de Microsoft offrent à l'instar des EJB de Sun un environnement d'exécution distribué, sécurisé, transactionnel et persistant pour des exécutions de composants .NET écrits en C#, Visual Basic .NET ou tout autre langage ayant un interpréteur `CLR`[9].

2.5.1 L'architecture COM+

COM+ est une spécification propriétaire de Microsoft. Par ailleurs, comme pour Sun J2EE/EJB, les environnements d'exécution .NET/COM+ peuvent être implémentés par d'autres fournisseurs que Microsoft. A l'heure actuelle, seul Microsoft propose une implémentation de son architecture.

L'architecture COM+ reprend les éléments suivants:

- *.NET Remoting (HTTP et TCP)* et DCOM permettent l'invocation de méthodes distantes pour les composants COM+.
- *System Directory Services* (SDS) permet l'utilisation d'un annuaire de noms pour trouver un composant COM+.
- *ADO.NET* assure la gestion des connexions à un service de persistance de données.
- *DTC COM+* assure la gestion des transactions ACID entre composants COM+ distribués.

2.5.2 Critiques

MS COM+ offre des outils similaires à l'architecture EJB et à la norme CORBA. Ainsi DCOM et *.NET Remoting* correspondent au protocole d'invocation distante de CORBA et EJB. SDS, quant à lui, est le pendant du

9. La CLR constitue dans le monde .NET Microsoft l'équivalent de la JVM Java de Sun

service d'annuaire CORBA et JNDI. Pour finir, DTC correspond au service de transactions de CORBA et à JTS.

Cependant .NET a le désavantage de ne pas être interopérable avec CORBA et EJB. Par ailleurs, les composants .NET présentent à l'instar des composants EJB le même inconvénient en terme d'utilisation des ressources.

Pour terminer, une critique générale de .NET et J2EE peut concerner le fait que ces deux frameworks facilitent énormément l'implémentation et le déploiement d'architecture quatre tiers (Présentation - Services - Business Object - Persistance) et peuvent, par ce biais, inciter les concepteurs d'architectures distribuées à uniquement se limiter à ce type d'architecture. Cette utilisation systématique du quatre tiers pourrait être réductrice en terme de créativité lors de la phase de conception et éventuellement en terme de l'adéquation de la solution informatique envisagée pour résoudre le problème rencontré.

2.6 Les Services Web

Selon l'Agence Wallonne des Télécommunications, les Services Web peuvent être définis de la manière suivante: *Le Service Web peut être compris comme une application exécutable disponible "en self-service" pour être utilisée par des clients (entreprises, applications, etc). Le Service Web est ainsi "exposé" sur le Web, avec la description de son fonctionnement et des paramètres dont il a besoin pour fonctionner. L'énorme avantage est de pouvoir utiliser un Service Web développé en Java sous Unix, dans une application tournant en Visual Basic dans un environnement .NET de Microsoft, puisque l'ensemble du dialogue se fera via des standards XML. Grâce à cette capacité à permettre le dialogue en applications hétérogènes, les Services Web vont jouer un rôle majeur pour l'intéropérabilité des applications et l'intégration de solutions.* [dT03]

Les Services Web ont été spécifiés par le W3C [Cond] et leur utilisation se fonde sur l'utilisation de plusieurs spécifications de protocoles du W3C et d'outils extérieurs. Ces protocoles sont le *Simple Object Access Protocol*, le *Web Services Description Language* et l'*Extensible Stylesheet Language Transformations*. L'utilisation des Services Web est facilitée grâce à l'*Universal Description, Discovery and Integration* défini par UDDI.org. Cet outil offre un registre d'informations catégorisant les Services Web.

2.6.1 Simple Object Access Protocol

Cette section constitue la synthèse des références suivantes: [dT03, Cond].

Proposé par le W3C [Cond], le *Simple Object Access Protocol* (SOAP) est un protocole d'appel de procédure distante (RPC) qui se base sur des standards établis tels que XML et HTTP. De part la nature des composants qui le constituent, SOAP est portable et interopérable. En effet, son fonctionnement est basé sur des éléments qui constituent actuellement l'internet (serveur HTTP, XML, etc.) et permet ainsi de passer au travers de *firewalls* sans paramètrage particulier et d'accéder à des services d'applications distribuées en réseau.

Les messages XML échangés en SOAP sont constitués de trois parties : une enveloppe qui définit ce qu'il y a dans le message et comment le délivrer ainsi que les données et les conventions de retour de résultat.

Comme son format de message de communication repose sur du texte au format XML, il lui est possible de collaborer avec des composants hétérogènes tels que CORBA, .NET ou J2EE écrits dans des langages couramment utilisés tels que Cobol, Java, C++, C#, Perl, etc.

2.6.2 Web Services Description Language

Cette section constitue la synthèse des références suivantes: [dT03, Cond].

Le *Web Services Description Language* (WSDL) est un langage permettant de décrire un service afin de permettre à deux Services Web voulant communiquer de se comprendre. A cette fin, pour un service, il définit le moyen de contacter le service, l'identité du service et le format (sous forme de DTD [10] XML) des messages entrants et sortants du service.

Par ailleurs, WSDL est indépendant des protocoles. En effet, WSDL spécifie comment utiliser SOAP, HTTP ou MIME comme couche transport des messages mais il reste possible d'utiliser des protocoles de *middlewares* existants tels que IIOP, RMI ou *Microsoft Message Queuing*.

2.6.3 Extensible Stylesheet Language Transformations

Cette section constitue la synthèse des références suivantes: [Conb, Conc].

L'*Extensible Stylesheet Language Transformations* (XSLT) est le langage de transformation de messages XML ayant des DTD différentes. Il est tout aussi possible de transformer un message XML en HTML ou en un autre message XML respectant une autre DTD. XSLT peut se montrer dès lors très intéressant pour faciliter la collaboration entre Services Web dont les formats de messages sont différents.

10. La Document Type Declaration (DTD) définit la grammaire d'un document XML.

2.6.4 Universal Description, Discovery and Integration

Cette section constitue la synthèse des références suivantes: [dT03, UDD].

L'*Universal Description, Discovery and Integration* (UDDI) est un standard récent (2000) qui propose un outil d'appel dynamique à des services. Cet appel se fait au moyen d'un registre d'information qui catégorise les Services Web proposés par des fournisseurs sur internet.

Dans ce registre, les fournisseurs de services inscriront la description en WSDL des interfaces d'accès à leur composant, les adresses pour accéder à leurs services, les droits d'accès,etc.

Ainsi UDDI permet aux fournisseurs de services d'enregistrer les specifications de leurs Services Web de manière à ce que des consommateurs puissent aisément les localiser et les utiliser au moyen de SOAP.

2.6.5 Critiques

Les services Web ont l'avantage sur d'autres technologies de distribution de pouvoir être déployés facilement dans des structures déjà en place. En effet, leur utilisation et leur déploiement passent souvent par l'utilisation d'un serveur HTTP doté des extensions adéquates. Cette utilisation fréquente, bien que non obligatoire, du protocole HTTP est liée au fait que l'usage du protocole HTTP et de XML ne pose en général aucun problème avec les configurations des *firewalls* existants. L'utilisation de standards tels que HTTP et XML ne peut dès lors qu'inciter à l'utilisation des Services Web.

Cependant, il faut se montrer prudent. En effet, certains protocoles et outils annexes utilisés (UDDI, SOAP et WSDL) n'ont pas encore montré toute leur maturité (UDDI ne date que de 2000) et sont encore en évolution (la sécurité est en cours d'étude). De même les performances du parsing XML ne font pas l'unanimité. Il faut aussi émettre une réserve quant à l'effet de mode suscité par les Services Web. En effet, ils sont considérés, à tort ou à raison, comme la nouvelle 'Silver Bullet' capable de résoudre beaucoup de problèmes dans la conception des systèmes distribués. De plus, les Services Web n'apportent rien de nouveau par rapport à des technologies plus anciennes (CORBA, EJB, etc.) si ce n'est l'utilisation de HTTP et XML. Pour ces raisons (jeunesse, pas de réelle nouveauté, mauvaises performances liées au parsing XML, etc.), il nous semble important d'être prudent dans l'adoption inconditionnelle de cette technologie.

2.7 Les systèmes Peer-to-Peer et GRID

Cette section constitue la synthèse des références suivantes: [tpWG, Shi00, Wil02, BGKS02, Ora01, Edw02, FKT01].

La plupart des systèmes distribués fonctionnent dans des environnements contrôlés, relativement stables et restreints. Ces environnements sont contrôlés et restreints car les diverses parties du système sont gérées et maintenues par une seule entité de gestion. Ils sont stables car même s'ils sont prévus pour être tolérants aux pannes, les événements qui provoquent ces pannes sont censés rester marginaux. Ces caractéristiques se retrouvent souvent dans les systèmes informatiques internes de la plupart des institutions et organisations. Il existe cependant deux classes de systèmes distribués travaillant dans des environnements plus larges, moins contrôlés et moins stables: les systèmes Peer-to-Peer (P2P) et GRID.

L'utilisation massive de l'Internet par les entreprises, les institutions publiques et les particuliers, a entraîné la création de ces nouvelles classes de systèmes distribués. Les systèmes P2P et GRID sont des systèmes distribués constitués d'hôtes ou de groupes d'hôtes connectés via des connexions sur de longues distances, souvent au travers de réseaux publics ou semi-publics tel l'Internet. Alors que les systèmes d'information distribués classiques sont conçus pour fonctionner sur un réseau d'hôtes pleinement contrôlés, les systèmes P2P et GRID le sont pour un réseau ouvert, dont les hôtes ne sont pas contrôlés, ou du moins faiblement contrôlés.

2.7.1 Les systèmes Peer-to-Peer

Si dès sa création, l'Internet avait pour objectif de rassembler des systèmes indépendants les uns des autres sans gestion centralisée, les premiers services proposés (FTP, email, Telnet, Gopher, etc.) furent conçus pour fonctionner dans un réseau stable. Ces premiers protocoles applicatifs se basent sur une disponibilité permanente des prestataires de services. Ils utilisent souvent le système de résolution de nom de domaine pour rechercher les prestataires de services alors que plusieurs jours y sont parfois nécessaires pour la mise à jour de l'information.

Avec l'explosion de la micro-informatique et l'évolution des technologies de communication, de plus en plus d'ordinateurs se sont connectés à l'Internet de façon ponctuelle, pour des durées plus ou moins courtes. Devant l'imminence du manque d'adresses réseaux pour toutes ces machines aux connexions éphémères, des systèmes d'attributions dynamiques et temporaires d'adresses ont vu le jour; avec pour résultat que de nombreux ordinateurs connectés se voient, aujourd'hui, dans l'impossibilité de fournir des services de qualité sur l'Internet. Il est difficile d'imaginer, par exemple, un serveur FTP ou un serveur WEB disponible en permanence sur une connexion de ce type.

Pourtant, il n'y a souvent que le mode de connexion de ces machines qui les empêche de fournir des services. De nos jours, la plupart des ordinateurs domestiques ont, par exemple, autant de capacité de calcul et de stockage que les premiers serveurs de l'Internet. Même s'il ne s'agit pas de

leur principale fonctionnalité, ces machines pourraient également être utilisées à des fins de services de l'Internet. D'autant plus, que bien souvent, une large part des ressources de ces machines ne se voit pas exploitée pleinement par leurs utilisateurs quotidiens. Les systèmes Peer-to-Peer (P2P) visent à l'utilisation de ces ressources via la spécification et l'implémentation de nouveaux services et protocoles. Le P2P se base essentiellement sur le partage de services par les machines situées au bord de l'Internet, sur les bureaux des utilisateurs finaux.

Les systèmes P2P les plus connus actuellement sont destinés aux utilisateurs privés avec des applications d'échange de fichiers (Gnutella, Freenet et Napster), de messagerie instantanée (ICQ et Jabber) ou de calculs distribués (Distributed.net, SetiHome). Mais certaines entreprises utilisent également des produits P2P pour améliorer leur productivité (messagerie instantanée chez Intel et IBM). Pour les entreprises, Sun a d'ailleurs développé son framework JXTA qui devrait permettre des implémentations rapides de produits basés sur le concept du P2P.

Le partage de ressources via les technologies P2P n'est cependant pas exempt de défauts. En effet, les systèmes actuels ne fournissent aucun mécanisme de contrôle des accès aux ressources partagées ou, tout au plus, fournissent-elles un contrôle centralisé en un point fixe du système. Les systèmes de partage de fichiers utilisent, par exemple, un mode de publication unique; les ressources sont toujours offertes à l'ensemble des utilisateurs. Les systèmes de calculs distribués, quant à eux, permettent le partage de ressources (temps de calcul ou mémoire), mais reposent sur un serveur central de coordination.

2.7.2 Les systèmes GRID

Pour des entreprises ou des institutions publiques, la technologie P2P peut servir à exploiter au maximum leurs ressources informatiques et favoriser la communication entre les membres du personnel. Cependant, les défauts de la technologie la rendent peu sûre pour une publication de services ou d'informations plus large, soumise à des conditions d'accès telles que le paiement, le respect de quotas d'utilisation, ou simplement la confidentialité. Or pour une réelle utilisation professionnelle, un mécanisme de contrôle du mode de publication s'avère nécessaire. C'est à ce besoin de partage des plus contrôlé que répondent les systèmes GRID.

Les systèmes GRID permettent le partage de ressources entre hôtes répartis au sein de plusieurs systèmes d'informations différents. Les différents systèmes d'informations participants à un système GRID fournissent des ressources et agissent ensemble comme un système global. Pour former un système GRID, ses participants fonctionnent sans centralisation de leur contrôle et de leur gestion. La gestion du système comme un tout est assurée par des mécanismes distribués et ne repose pas sur une forme de centralisation.

Le principal objectif des systèmes GRID est de pouvoir mettre en relation plusieurs systèmes de taille conséquente. Un système GRID de calcul distribué permet de connecter plusieurs supercalculateurs, délivrant ainsi une puissance de calcul hors de portée des différents fournisseurs de services sans le passage par un système GRID. Les systèmes GRID visent la mise en commun des ressources disponibles au sein d'organisations et non plus au sein d'un groupe d'utilisateurs de machines de bureau.

Dans les solutions techniques utilisées, les systèmes GRID et P2P se ressemblent beaucoup, et la frontière entre les deux reste floue. Par rapport au P2P, les systèmes GRID travaillent dans un environnement plus stable, leurs participants types sont de larges organisations bénéficiant de connexions performantes et de matériel de haut niveau. Un système P2P vise à partager les ressources fournies pas ses utilisateurs, les utilisateurs de services y sont également fournisseurs. Dans un système GRID, la distinction entre utilisateurs et fournisseurs reste plus importante, les utilisateurs y sont souvent des membres des organisations participantes, qui, elles, sont fournisseurs des services.

2.8 Conclusion

Un système distribué est un système d'information constitué de composants répartis généralement sur plusieurs *hôtes* interconnectés entre eux au moyen d'un *midlleware*. Le rôle du *middleware* est de fournir une couche d'abstraction assurant une communication plus facile et plus "transparente" entre les composants du système en évitant au développeur de gérer les aspects techniques liés à leur distribution. Si la complexité des systèmes distribués est accrue par rapport aux systèmes centralisés, ils bénéficient, cependant, d'une plus grande robustesse et d'une plus grande fiabilité.

Il existe de nombreux *midllewares*, le plus ancien étant certainement la norme CORBA spécifiée par l'OMG. Malgré des avantages, tels que l'interopérabilité entre langage et la possibilité de l'utiliser au-dessus de systèmes existants, CORBA souffre de sa complétude. En effet, CORBA offre une grande généricité et un nombre important de fonctionnalités mais ces atouts peuvent amener une complexité inutile à la réalisation de tâches simples. Actuellement, d'autres *midllewares* existent tels que Sun J2EE ,Microsoft .NET ou les Services Web, offrant chacun leurs avantages et leurs inconvénients.

Si les systèmes distribués fonctionnent généralement dans des environnements contrôlés, l'accroissement des communications inter-organisations et entre les ordinateurs de bureau a permis l'émergence de systèmes distribués répartis dans des environnements nettement plus "chaotiques". Par rapport aux systèmes distribués "classiques", les systèmes de ce type ne voient pas, en général, leurs ressources contrôlées par un système de gestion centralisée.

Les systèmes Peer-to-Peer permettent ainsi de former des systèmes distribués dont les *hôtes* sont constitués des machines de bureau des utilisateurs du système, les mêmes ressources permettent l'accès au système et réalisent aussi les traitements. Les systèmes GRID visent, quant à eux, la constitution d'un seul système distribué unifié en procédant à la mise en commun des systèmes d'informations de plusieurs organisations sans qu'il n'y ait d'administration globale de l'ensemble ainsi formé.

Chapitre 3

Le Projet FEDERGEN

3.1 Introduction

Ce chapitre a pour objectif de présenter, de manière concise et non exhaustive, le travail qui a été réalisé par Laurent Debaisieux et Fernando Desouza dans le cadre de leur mémoire [DD03] portant sur le Projet FEDERGEN. Ce projet visait le partage de ressources bioinformatiques hétérogènes. De leur mémoire, nous n'avons retenu que les éléments clés qui nous ont servi de point de départ pour notre réflexion.

3.2 Un médiateur de banques de données génétiques

3.2.1 Architecture générale

Pour tenter de répondre à une série d'exigences des bioinformaticiens, Laurent Debaisieux et Fernando Desouza ont proposé une architecture permettant de rendre l'utilisation de la norme BSA (voir section 1.4 page 9) plus simple et flexible pour les utilisateurs et les développeurs de ressources bioinformatiques.

Selon la norme BSA, les fonctionnalités des *legacy systems* sont encapsulées dans les `AnalysisServices`. Par souci de simplicité, les auteurs ont décidé de nommer *wrappers* les `AnalysisServices` et les éléments qui y seront adjoints par nécessité dans leur architecture. Il est important de rappeler que ces *wrappers* peuvent être physiquement hébergés sur plusieurs hôtes et qu'un hôte peut héberger plusieurs *wrappers*.

De manière à couvrir les points non définis par la norme BSA (la localisation d'un `AnalysisService`, l'invocation de l'`AnalysisInstance` implémentant le service, la fourniture des paramètres d'entrée nécessaire à l'exécution du service et la récupération du résultat), Laurent Debaisieux et Fernando Desouza ont ajouté un élément au système "client-wrapper" (voir figure 3.1):

le médiateur (voir figure 3.2).

L'utilisation de ce médiateur offre de multiples possibilités telles que proposer des services à valeur ajoutée, gérer les appels aux *wrappers* ainsi que gérer les accès et la performance. Par exemple, le médiateur pourrait proposer un *wrapper* "virtuel" consistant en l'agrégation de l'exécution de plusieurs services au moyen d'un langage de script. Ou bien, le médiateur restreindra l'accès à tel ou tel *wrapper* en fonction du département de recherche du client et moyennant authentification de sa part.

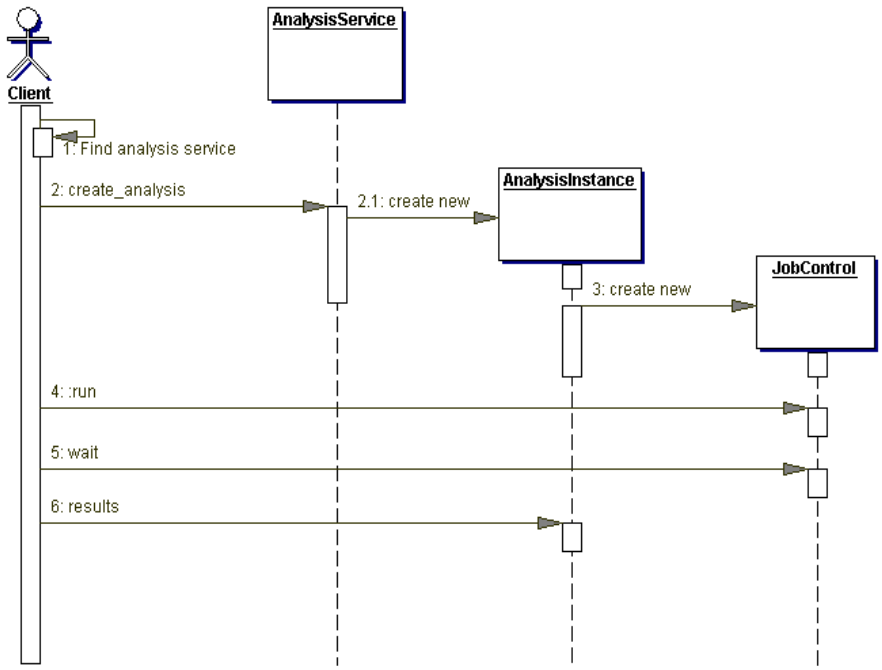


FIG. 3.1 – *Diagramme de séquence de l'utilisation d'un wrapper tel que défini par la norme BSA*

Comme le précisent les auteurs, le médiateur est amené à jouer un rôle central:

"Ce médiateur sera le point de contact unique des wrappers lorsque ceux-ci voudront intégrer le système. De même, le médiateur sera le point de contact unique des clients désirant adresser une requête au système. Le médiateur a donc un rôle central dans notre architecture. Quand un client soumet une requête au médiateur, celui-ci, connaissant tous les AnalysisServices disponibles (et leur localisation), peut la transmettre à l'AnalysisService ad hoc. Les clients adressent leurs requêtes au médiateur conformément à la norme BSA car le médiateur est "vu" comme un ensemble d'AnalysisServices (dans un premier temps)". [DD03]

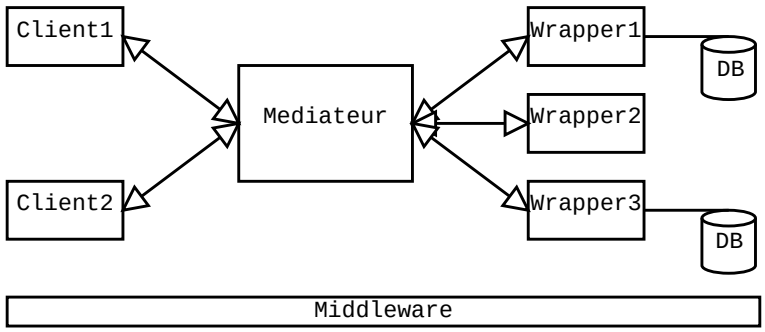


FIG. 3.2 – *Architecture générale à un médiateur proposée par [DD03]*

3.2.2 Ajouts à la norme BSA

Comme annoncé dans la description de leur architecture générale, Laurent Debaisieux et Fernando Desouza ont dû adjoindre à la norme BSA une série de mécanismes permettant le fonctionnement du médiateur:

- Un mécanisme permettant à un *wrapper* de se déclarer auprès d'un médiateur. Cette déclaration se base sur le protocole suivant: d'abord, le *wrapper* signale son existence auprès du médiateur grâce au message `hello`. Ensuite, le *wrapper* signale au médiateur, au moyen d'une série de messages `describe`, les propriétés du service dont il est le fournisseur. Ces différentes propriétés comprennent l'`AnalysisType` ainsi que les types de données des paramètres d'entrée et de résultat.
- Un mécanisme de distribution de l'IOR CORBA [1] du médiateur aux divers clients potentiels: les auteurs ont proposé de publier l'IOR du médiateur au moyen d'une page Web accessible à tous.
- Un mécanisme de centralisation de l'information permettant de contacter les services disponibles: l'*IOR Repository*. Cet *IOR Repository*, propre au médiateur, contiendra les IOR des *wrappers* connus de celui-ci.
- Pour terminer, un mécanisme qui offre la possibilité aux clients d'un médiateur d'obtenir la liste des *wrappers* actifs disponibles.

3.2.3 Critères de comparaison de *wrappers*

Pour rappel, la norme `BSA` spécifie, dans l'`AnalysisType`, différents attributs caractérisant un `AnalysisService`. Ces attributs sont:

- L'attribut `Type`: cet attribut permet de classer les analyses en fonction d'une classification préétablie.

1. L'IOR est la représentation standardisée des références d'objets distants dans la norme CORBA.

- L'attribut `Name`: cet attribut est utilisé pour nommer l'analyse dans le système.
- L'attribut `Supplier`: cet attribut identifie le responsable de l'implémentation de l'`AnalysisService`.
- L'attribut `Version`: cet attribut spécifie le numéro de version de l'implémentation.
- L'attribut `Installation`: cet attribut précise la localisation d'une installation d'un `AnalysisService`.
- L'attribut `Description`: cet attribut décrit le service rendu.

Remarque: La norme `BSA` prévoit que cette liste puisse être éventuellement étendue en fonction des besoins.

Pour être en mesure de comparer des `AnalysisServices`, Laurent Debaisieux et Fernando Desouza ont insisté sur la nécessité d'uniformiser et de standardiser la description offerte par les `AnalysisServices` au moyen de leur `AnalysisType`. Dans cette optique, ils ont ajouté dans leur architecture le *Master Definition Repository* (voir section 3.2.4).

Cependant, un problème paraissait subsister: comment déterminer si deux services sont identiques?

Afin d'y remédier, les auteurs ont émis la proposition suivante : *"En d'autres termes, quels sont les attributs de l'AnalysisType à prendre en compte pour déterminer l' " identifiant ". La question de la composition de l'identifiant d'un AnalysisService reste posée, mais nous pensons qu'il est préférable que cette identification soit la plus précise possible, ce qui permettra par après de faire le choix d'un AnalysisService en fonction de différents critères (performance, etc.)".* [DD03]

3.2.4 Master Definition Repository

Chaque médiateur est doté d'un *IOR repository* contenant les IOR des *wrappers* (`AnalysisServices`) qui lui sont directement connectés. En plus de ce repository, un repository central contenant tous les `AnalysisTypes` (définition du type d'analyse) ainsi que le format de leurs paramètres d'entrée et de résultat doit être inclus dans le système. Ce *Master Definition Repository* (MDR) fonctionne lors de l'inscription auprès du médiateur d'un nouveau *wrapper*. Si le type de ce dernier n'est pas encore connu, il sera ajouté au MDR.

3.2.5 Les *proxies*

Par rapport à la norme BSA originale, Laurent Debaisieux et Fernando Desouza ont ajouté un élément fondamental qui est le médiateur. L'ajout de ce médiateur a pour objectif de rendre la localisation des *wrappers* transparente pour l'utilisateur. Les requêtes envoyées par un client ne devront

plus être adressées directement au *wrapper* mais au médiateur. Ce découplage du client et du *wrapper* permettra ultérieurement l'ajout de services à plus-value (sécurité, etc.) par rapport à l'utilisation directe des *wrappers* BSA. Il est important de préciser que le client s'adressera au médiateur en respectant la norme BSA lorsqu'il désirera effectuer un appel à un service. Pour parvenir à leur objectif de découplage complet du client et du *wrapper*, les auteurs ont proposé l'ajout d'un mécanisme de *proxy* [GHJV94] aux `Analysis Services`. Ces *proxies* basés sur le *Design Pattern* correspondant [GHJV94] est en quelque sorte une image au sein du médiateur d'un *wrapper* réel. Ainsi, un *proxy* d'`AnalysisService` sera créé pour chaque *wrapper* enregistré auprès du médiateur et son IOR sera stocké dans l'*IOR Repository* du médiateur. Ce *proxy* sera l'entité avec laquelle le client dialoguera en BSA (`create_analysis`, etc.) lors de ses appels. Pour utiliser un *wrapper*, le client demandera au médiateur l'IOR du *wrapper* qu'il souhaite utiliser et le médiateur lui renverra l'IOR correspondant au *proxy* du *wrapper*.

3.3 Une Fédération de médiateurs

Pour pouvoir répondre à l'exigence suivante: "Une société privée doit pouvoir utiliser l'architecture afin de profiter des services proposés universellement tout en rendant l'accès à certains services internes confidentiels voir payants", Laurent Debaisieux et Fernando Desouza ont modifié leur architecture pour permettre à plusieurs médiateurs de fonctionner en une fédération de médiateurs partageant des `AnalysisServices` spécifiques.

3.3.1 Architecture globale de la fédération de médiateurs

Dans cette nouvelle architecture (voir figure 3.3), chaque médiateur, lorsqu'il rejoint la fédération, est libre de communiquer aux autres médiateurs les *wrappers* qu'il connaît. Ainsi, chaque fois qu'un *wrapper* s'enregistrera auprès d'un médiateur, celui-ci sera libre ou non de le signaler au reste de la fédération. Pour parvenir à atteindre cet objectif, Laurent Debaisieux et Fernando Desouza ont décidé d'ajouter un nouvel élément à leur architecture: *"...nous pensons que chaque médiateur doit avoir un carnet d'adresses précisant qui ils doivent prévenir lorsqu'ils sont avertis de l'arrivée d'un nouveau wrapper ou de la modification d'un IOR."* [DD03] Ils ont appelé ce carnet d'adresses *Friend Office IOR Repository*.

3.3.2 Le *chaînage des proxies*

Dans l'optique de contrôle des accès et du découplage des clients et des *wrappers*, Laurent Debaisieux et Fernando Desouza ont proposé que les appels entre médiateurs passent également par des *proxies*.

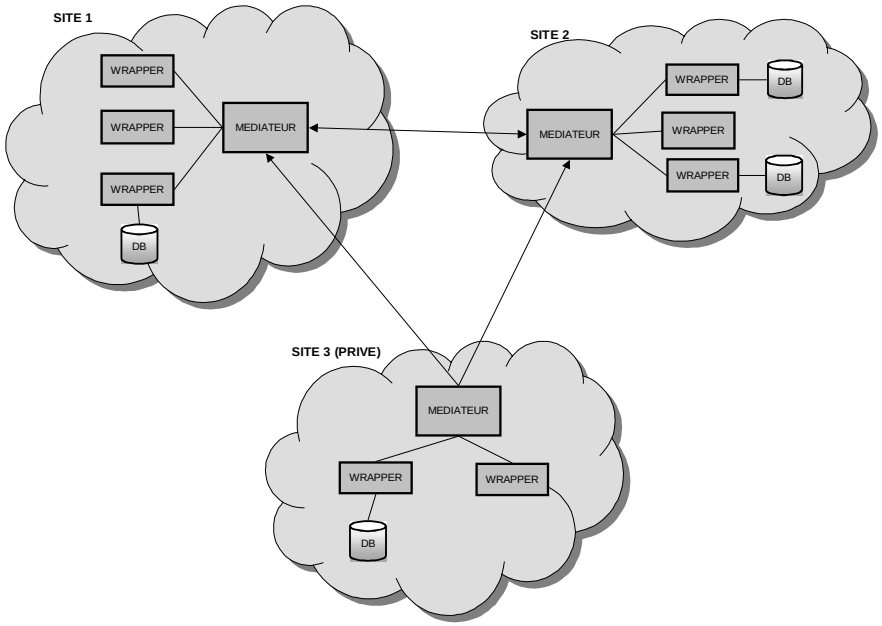


FIG. 3.3 – *Architecture générale à plusieurs médiateurs proposée par [DD03]*

Dans l'exemple (voir figure 3.4) issu du mémoire [DD03], illustrant le cas d'un appel entre un client et un *wrapper* séparés par deux médiateurs, les deux *proxies* (situés chacun dans leur médiateur respectif) relaieront les appels au *wrapper* de la manière suivante:

1. Après avoir consulté le médiateur2 pour obtenir la liste des services disponibles, le client désire adresser un `create_analysis()` concernant un service idw_1 au médiateur2. Il contacte le médiateur2 qui lui renvoie l'IOR de son *proxy* (w_1'') du service idw_1. Ensuite, le client effectue un `create_analysis()` sur le *proxy* du service idw_1 auprès du médiateur2.
2. Suite au `create_analysis()` du client, l'`AnalysisInstance` créé par le *proxy* consulte l'*IOR Repository* du médiateur2 et constate qu'il peut répondre à l'appel du client sur le service idw_1 en relayant (*forward*) l'appel vers l'IOR du *proxy* du service idw_1 du médiateur1 (w_1').
3. Le *proxy* du service idw_1 du médiateur2 répercute alors le `create_analysis()` à l'IOR w_1' du *proxy* du service idw_1 du médiateur1.
4. Le *proxy* du service idw_1 du médiateur1 reçoit un `create_analysis()` concernant le service idw_1. Ensuite, l'`AnalysisInstance` créé par le *proxy* du médiateur1 consulte l'IOR Repository et constate qu'il peut répondre à l'appel sur le service idw_1 en le relayant (*forward*) vers l'IOR w_1.

5. Le *proxy* du service idw_1 du médiateur1 répercute le `create_analysis()` à l'IOR w_1 qui arrive au *wrapper* qui encapsule le service. Ce dernier effectuera réellement l'analyse.
6. Les résultats quant à eux seront propagés du *wrapper* vers le médiateur2 qui les renvoie au client, en passant par le médiateur1.

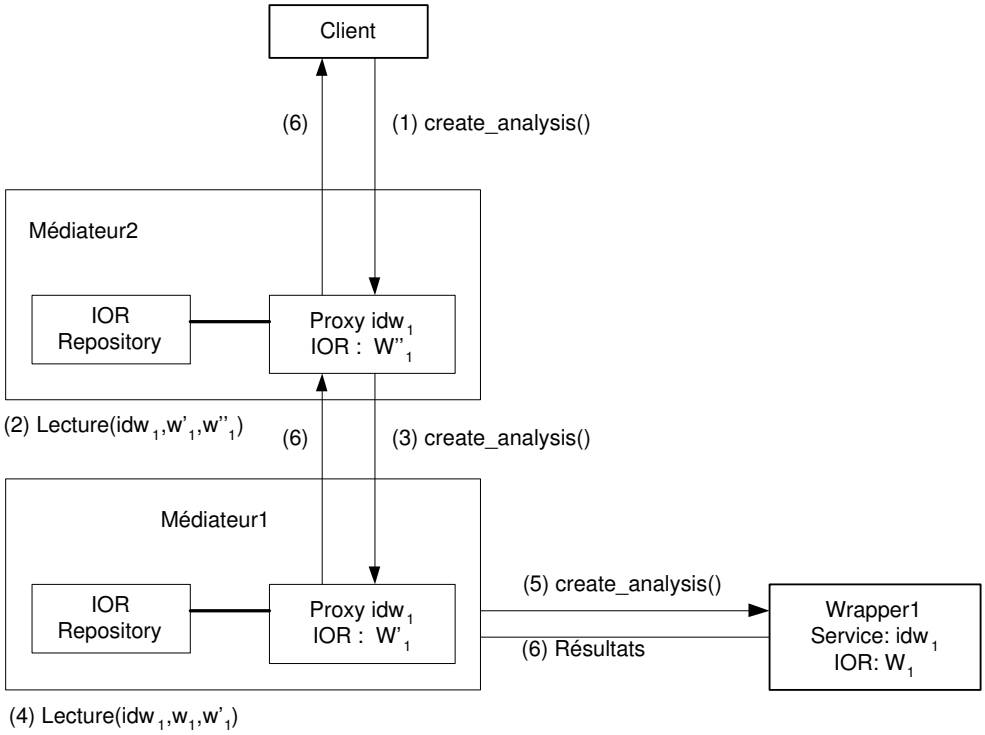


FIG. 3.4 – *Exemple de* chaînage de proxies *[DD03]*

Ce processus s'applique quel que soit le nombre de médiateurs.

On peut observer grâce à cet exemple qu'un *wrapper* peut aisément se faire connaître à toute la fédération et un client peut adresser une requête à un médiateur qui se chargera de répercuter cette requête à travers la fédération jusqu'au destinataire final capable d'exécuter cette requête.

3.4 Critiques

Suite à cette présentation non exhaustive des principes fondamentaux de l'architecture élaborée par Laurent Debaisieux et Fernando Desouza dans le cadre de leur mémoire [DD03], nous aimerions émettre quelques critiques et propositions d'amélioration.

Tout d'abord, le nombre de *repositories* différents (*IOR Repository*,*Master Definition Repository*, *Friend Office Repository*) nécessite, bien que justifié par la démarche incrémentale de conception des auteurs, des raffinements pour permettre une meilleure adaptation au contexte distribué de la fédération de médiateurs. Pour cette raison, nous avons décidé de proposer dans notre analyse une solution alternative plus adaptée au contexte d'une fédération distribuée de médiateurs et qui permettra une distribution performante du catalogue des services (résistance à la montée en charge, sécurité, etc.) et intégrera les différents éléments répertoriés par les auteurs dans leurs différents *repository*.

Ensuite, Laurent Debaisieux et Fernando Desouza n'ont pas explicité de mécanisme précis du "routage" (*forward*) des appels aux *wrappers*. Ainsi, le cas de figure où un même *wrapper* est accessible par deux *chaînes de proxies* différentes dans la fédération peut poser problème et n'a pas non plus été abordé par les auteurs. De même, le prototype qu'ils ont implémenté valide uniquement l'architecture de la fédération de médiateurs dans le cas d'une communication *client - proxy médiateur - wrapper*. Pour ces raisons, nous nous proposons de définir une solution de "routage" des appels aux *wrappers* en fonction des *chaînes* disponibles dans la fédération et de valider notre solution au moyen d'un prototype. Il est également important de remarquer que la définition de l'identifiant d'un `AnalysisType` n'a pas été clairement définie par Laurent Debaisieux et Fernando Desouza ainsi que les éventuelles propriétés dynamiques d'un service (performance, etc.). Nous tenterons donc de spécifier ces éléments de manière plus complète.

Il faut noter que les mécanismes de sécurité tels que les droits d'accès aux *wrappers* et aux médiateurs par les clients n'ont pas été définis par les auteurs et méritent réflexion.

Par ailleurs, il nous semble également nécessaire d'évaluer la pertinence de l'utilisation de la norme BSA dans le cadre de la communication *Client - Médiateur*. En effet, dans ce cadre, le client est amené à utiliser d'autres éléments qui ne sont pas du ressort de la norme BSA (obtention de la liste de *wrappers* disponibles, authentification éventuelle, etc.). Ces éléments nous semblent suffisants pour remettre en question l'utilisation de la norme BSA dans le cadre de la communication entre le client et le médiateur.

Enfin, la conception d'une architecture utilisable et évolutive nous paraît nécessaire. A priori, une architecture en couche pourrait constituer un bon point de départ à notre réflexion.

3.5 Conclusion

Dans leur mémoire [DD03], Laurent Debaisieux et Fernando Desouza proposent une architecture d'accès à des services d'analyses bioinformatiques se basant sur le norme BSA (voir section 1.4 page 9). Lors d'une première

étape, ils ont proposé un système trois tiers, composé de clients, d'un médiateur central et de *wrappers*. Un *wrapper* est un composant logiciel qui assure la réalisation d'analyses scientifiques ainsi que diverses fonctionnalités. Les clients utilisent un médiateur pour découvrir les *wrappers*, y accéder et les utiliser. Toutes les interactions entre les *wrappers* et les clients sont également contrôlées au moyen d'un système de *proxies*, de manière transparente pour l'utilisateur. Les *wrappers* s'inscrivent auprès d'un médiateur pour annoncer leur disponibilité. En plus de la liste des *wrappers* actifs, un médiateur possède une certaine connaissance des services qu'il propose, ce qui lui permet de faciliter les recherches des clients mais aussi d'apporter d'éventuels services à valeur ajoutée; services se basant eux-même sur les *wrappers* inscrits auprès du médiateur.

Dans un second temps, les auteurs proposent d'élargir le système à une fédération de médiateurs interconnectés. Chaque médiateur présent dans cette fédération serait alors libre de partager les services des *wrappers* inscrits auprès de lui, avec d'autres membres de la fédération. De manière à assurer une découpe nette entre les médiateurs et leur garantir le contrôle des opérations tout en permettant le partage de *wrappers* entre médiateurs non directement connectés, les auteurs proposent d'utiliser des *chaînes de proxies* entre les clients et les *wrappers*. Ainsi, lorsqu'un client accède à un service connecté à un autre médiateur que le sien, une série de *proxies* sont créés dans chacun des médiateurs situés entre le sien et celui du *wrapper* qu'il désire utiliser.

L'architecture proposée, même si elle constitue une bonne base, souffre de quelques défauts. En effet, certains mécanismes n'ont pas été complètement spécifiés ou manquent de précision. Ainsi, l'architecture globale de la fédération de médiateurs n'a pas été entièrement définie. De même, certains aspects à prendre en compte dans le contexte d'un système distribué n'ont pas non plus été couverts; de même, les mécanismes de gestion des utilisateurs et de la sécurité sont tout simplement absents. C'est pour cette raison, que nous proposons de raffiner et de compléter l'architecture présentée par Laurent Debaisieux et Fernando Desouza de manière à en corriger les défauts et y apporter les éléments manquants.

Deuxième partie

Analyse

Chapitre 4

Méthodologie d'analyse

4.1 Cadre de réalisation

La conceptualisation et l'implémentation du système ont été effectuées durant notre stage de quatre mois dans l'unité de bioinformatique de l'Université Libre de Bruxelles (ULB). La réalisation du travail prend suite au mémoire réalisé par Laurent Debaisieux et Fernando Desouza [DD03] dans le cadre de leur licence à horaire décalé en informatique aux Facultés Universitaires Notre-Dame de la Paix à Namur (FUNDP). Notre analyse se base sur les éléments de leur travail disponibles dès Septembre 2002.

4.2 Objectifs

L'objectif de notre analyse est la conception d'une architecture et l'implémentation d'un prototype d'une fédération de partage de ressources bioinformatiques hétérogènes. Nous avons souhaité reprendre la conception du système depuis le début, comme s'il s'agissait d'un développement nouveau, tout en utilisant l'architecture proposée par [DD03] comme point de départ à notre réflexion. D'un point de vue fonctionnel, l'objectif principal du travail est un système de partage de ressources simple sur lequel doit pouvoir se greffer des fonctionnalités plus complexes. Nous avons donc décidé de définir les grandes lignes d'une architecture pour ce système. Cette architecture se veut la plus ouverte possible et la plus résistante à la montée en charge. Pour atteindre cet objectif, nous n'avons spécifié précisément que les mécanismes permettant le partage des ressources et les appels aux services. Les autres éléments intervenant dans l'architecture devront être encore raffinés. Conjointement, nous avons également décidé de réaliser notre prototype en implémentant ces parties pleinement spécifiées et un maximum d'éléments définis, même de manière succinte, dans notre architecture.

4.3 Méthodologie suivie

La méthodologie de développement que nous avons suivie s'inspire essentiellement de celle que nous avons appliquée lors du travail pratique que nous avons réalisé dans le cadre du cours "Laboratoire d'Ingénierie du Logiciel [INFO 2205]". Ce choix a été essentiellement guidé par le fait que cette méthodologie est la seule que nous connaissions et avions pratiqués auparavant.

Notre méthodologie se découpe en cinq phases: *Analyse des exigences*, *Conception Logique*, *Conception Physique*, *Implémentation* et, pour terminer une phase de *Critiques*. Les phases de conceptions logique et physique ont été suivies de réunions de validation avec notre promoteur et les membres de l'unité de bioinformatique de l'ULB.

La phase d'*Analyse des exigences* permet une mise à plat des exigences fonctionnelles et non fonctionnelles souhaitées pour l'application. Elle reprend les interviews et résumés d'interviews des membres de l'équipe de l'unité de bioinformatique de l'ULB que nous avons considérés comme nos *clients*. Suite à ces énoncés informels, nous avons réalisé des cas d'utilisation (Uses Cases) UML ainsi qu'un diagramme de classes purement conceptuel des concepts du domaine et leurs inter-relations. Ces cas d'utilisation et ce diagramme de classes ont pour objectif de convenir d'une terminologie entre *analystes* et *clients* mais aussi de valider notre compréhension des exigences de notre *client*.

La phase de *Conception Logique* quant à elle, est plus volumineuse. En partant des *exigences* recensées lors de la phase précédente nous avons décidé des méthodes de fonctionnements permettant de réaliser les différentes fonctionnalités requises par l'application. C'est lors de cette phase qu'est proposée l'architecture logique, ou architecture de fonctionnement, de l'application ainsi que les modèles servant à représenter ces données. Les mécanismes plus complexes nécessaires à certaines fonctionnalités sont affinés via une présentation de la dynamique des composants de l'architecture proposée.

Lors de la phase de *Conception Physique* nous avons défini les détails *plus physiques* nécessaires à l'implémentation de l'architecture produite lors de la phase de *Conception Logique*. Ainsi, lors de cette phase, nous avons déterminé la découpe physique des composants en terme d'hôtes. Nous avons également défini l'architecture physique de persistance (données conservées, moyens de les conserver, structure utilisée et dépendance avec les composants physiques).

Pendant la phase d'*Implémentation*, nous avons tenté d'implémenter un maximum de fonctionnalités de notre architecture tout en respectant au maximum nos spécifications logiques et physiques.

La phase de *Critiques* consiste en une suite de critiques a posteriori et de propositions de futures améliorations qu'il est possible d'apporter à notre architecture logique. Cette critique se base sur l'expérience que nous avons

acquise lors de notre analyse et des éventuels problèmes rencontrés lors de notre implémentation.

Chapitre 5

Analyse des exigences

5.1 Introduction

Ce chapitre a pour but de présenter les exigences fonctionnelles et non fonctionnelles des différents acteurs vis-à-vis du système. Tout au long de ce chapitre, notre démarche consiste à analyser les interviews des différents membres de l'unité de bioinformatique de l'IBMM et à déterminer, sur base de cette analyse, les cas d'utilisation (Use Cases) du futur système, les exigences non fonctionnelles et à présenter un diagramme de classes des principales notions du domaine d'application. Par ailleurs, il est important de signaler que nous avons procédé, lors de notre stage, à une validation de cette analyse par les membres de l'unité de bioinformatique de l'IBMM.

5.2 Résumé des différentes interviews

Cette section a pour objectif de regrouper et de résumer les diverses exigences formulées par des Bioinformaticiens au cours de nos différents entretiens.

- Nous sommes désireux d'une fédération de partage de services bioinformatiques qui respecte la norme BSA de l'OMG (voir section 1.4 page 9). En effet, cette norme est la seule qui définisse de manière générique la représentation en objet des séquences nucléiques et protéiques. Par ailleurs, elle fournit également une spécification complète permettant la conception de *wrappers* de services existants. La norme définit aussi les interactions à ces services "wrappés" au moyen de diagrammes de séquences. Par ailleurs, la norme BSA est tout à fait générique et n'impose pas l'utilisation de CORBA comme technique de distribution.
- Le choix de CORBA nous semble un choix judicieux. Il est plus facile d'y intégrer des composants BSA, bien que cette norme soit générique. De plus, l'European Bioinformatics Institute (EBI) [Ins03] est

un fervent défenseur de CORBA. Bien que souhaitable, l'utilisation de CORBA n'est pas obligatoire.

- La future fédération sera capable de migrer vers de nouvelles technologies de distribution et d'accès à des services telles que les Services Web [Cond], J2EE [Micb], .NET [Mica], etc.
- La fédération devra accepter le partage d'outils anciens (*legacy components*) et de composants développés entièrement en interne par des universités ou des partenaires privés.
- Les services bioinformatiques mis à disposition de la fédération sont constitués de services de séquençage [1], de services d'annotation [2], de services d'alignement [3], des documents et articles scientifiques, des accès à des banques de données de génomes, etc.
- La fédération devra permettre à un utilisateur normal d'utiliser un service bioinformatique qui convient le mieux à ses attentes. L'utilisateur devra pouvoir rechercher et trouver ce service bioinformatique, ceci de la manière la plus naturelle possible. L'outil de recherche du service le plus adéquat doit pouvoir répondre à la demande suivante : "Je veux réaliser un alignement entre les séquences A et B que je possède.
- L'utilisateur de services bioinformatiques, aura la possibilité de vérifier l'état de ses comptes, c'est-à-dire, connaître l'état des quotas de ressources qu'il a déjà utilisés ainsi que les quotas qu'il peut encore consommer.
- La fédération prendra en compte différents profils d'utilisation. Ces profils vont de l'étudiant en biologie moléculaire au Bioinformaticien en passant par des cliniciens et des chercheurs novices ou confirmés ainsi que par l'administrateur du système. Les compétences informatiques de ces différents utilisateurs vont nulles à très élevées. L'adaptation du système au profil de l'utilisateur ainsi qu'à son niveau de connaissance des technologies de l'information et des outils bioinformatiques est cruciale afin de faciliter son acceptation au sein de la communauté scientifique.
- La fédération permettra la gestion (création, suppression, modification, etc.) de groupes l'utilisateurs. Le système doit permettre le regroupement d'utilisateurs en fonction de leur laboratoire, besoins, appartenance à des groupes de travail, etc.
- La fédération permettra une gestion de droits d'accès aux services bioinformatiques disponibles. Ces droits d'accès seront fonction du profil

1. Séquençage : technique permettant de déterminer l'ordre des composants d'une macromolécule (les acides aminés (20 différents) d'une protéine, les nucléotides (4 différents) d'un acide nucléique, etc.). [GJ01b]

2. Annotation : commentaire portant sur une région de séquence (Nucléique ou protéique) déduite de données expérimentales ou de techniques prédictives. [GJ01b]

3. Alignement : technique superposant des séquences différentes à des fins de comparaison. [GJ01b]

de l'utilisateur et de ses diverses appartenances à des groupes d'utilisateurs. L'utilisateur pourra également avoir accès à des services particuliers indépendamment de tout groupe ou profil.

- Un utilisateur qui partage un service doit pouvoir choisir les profils, les groupes et les utilisateurs qui y ont accès. Une autorisation de l'administrateur du système devra être requise.
- Les partenaires de la fédération seront variés. Nous trouverons des universités, des entreprises privées, des organismes d'état ou des organisations non gouvernementales.
- En raison de la diversité des futurs partenaires de la fédération, il sera fondamental de leur garantir une sécurité lors de l'utilisation des services qu'ils partageront au sein de la fédération. Ainsi, une entreprise privée qui collabore avec une université ne souhaite pas forcément que ces services soient partagés avec les autres membres de la fédération. Une gestion de la diffusion des services et de la sécurité entre les membres de la fédération est essentielle.
- La sécurité des paramètres d'analyse d'un chercheur ainsi que celle des résultats devront être garanties. On veillera à ce que seul le service et son client se comprennent. On veillera dès lors à éviter l'interception de ces données par un tiers présent au sein de la fédération.
- La fédération utilisera un modèle de description des services disponibles à la fois standardisé et parfaitement défini. De cette manière, il sera possible de générer automatiquement des interfaces homme-machine qui s'adaptent à l'utilisateur en fonction des services qu'il a choisi d'utiliser.
- Le résultat de l'utilisation d'un service sera complètement "protocolé". Un chercheur doit connaître les conditions dans lesquelles ses résultats ont été obtenus. L'utilisateur devra connaître les versions des programmes (qui évoluent régulièrement suite à des améliorations ou corrections), les versions de banques de génomes sur lesquelles se basent ces programmes et tout autre paramètre susceptible d'influencer le résultat de l'exécution du service et sa reproductibilité.
- Sur base de l'utilisation d'un service et de son protocole de résultat que nous appellerons "session", un utilisateur doit être en mesure de sauvegarder cette session mais aussi de l'exécuter à nouveau dans des conditions aussi proches que possible des conditions d'exécution initiales.
- Un utilisateur devra être en mesure de partager ses sessions d'utilisation avec d'autres utilisateurs (collègues de laboratoires, supérieurs hiérarchiques, etc.) du système.
- Un utilisateur devra bénéficier d'outils lui permettant d'enchaîner, à la manière d'un traitement "batch", une suite d'exécutions de services. Le résultat d'un service constitue les données d'entrées du service sui-

vant et ainsi de suite. Il est évident que des traitements intermédiaires devront être permis pour réaliser d'éventuels changements de formats des données entre chaque étape. Un langage de scripts facile d'accès et d'utilisation mais complet au niveau de ses possibilités (ex: expressions régulières, exécution d'appels en parallèle, point de synchronisation pour appels parallèles, etc.) serait un plus pour l'utilisateur avancé.

Par exemple, un biologiste moléculaire souhaiterait réaliser la suite d'opérations suivantes: récupérer deux gènes particuliers présents dans le génome de deux espèces et stockés dans deux banques de génomes différentes; et ensuite, comparer les deux séquences au moyen d'un service d'alignement pour connaître la "similarité" entre les deux gènes. Actuellement, le scientifique doit se rendre sur le site Web des deux banques et faire proprement des copier-coller des séquences concernant les gênes qu'il veut étudier. Ensuite, il doit éventuellement convertir les formats des deux séquences dans le format accepté par le logiciel d'alignement. Pour terminer, il peut exécuter son alignement et analyser le résultat de celui-ci.

S'il disposait d'un langage de scripts, il pourrait écrire préalablement un script demandant d'exécuter en parallèle les recherches des deux gènes dans les deux banques de génomes. Il préciserait ensuite que l'interpréteur de scripts doit attendre les résultats de ces deux recherches au moyen d'un point de synchronisation. Ensuite, il pourrait ajouter des instructions de manipulation de résultat au moyen des fonctions du langage de scripts. Puis, il spécifierait le service d'alignement à utiliser pour son analyse. Pour finir, il démarrerait l'exécution de son script d'analyse.

- Le système devra être dynamique. Il permettra des ajouts et des suppressions de membres de la fédération de manière rapide et dynamique, même si, dans la pratique, la topologie de la fédération restera assez statique.
- Le système devra réagir de manière dynamique aux ajouts et suppressions de services disponibles au sein de la fédération ainsi qu'à leur indisponibilité temporaire.
- Les services publiés par les différents membres de la fédération devront être accessibles par un maximum d'utilisateurs.
- Le système devra bénéficier d'une résistance maximale à la montée en charge. On ne connaît pas le nombre de noeuds a priori mais la fédération serait susceptible de contenir au moins une centaine de partenaires institutionnels.
- La fédération devra maximiser la "qualité de service" de l'utilisation des services. Elle offrira un accès optimal pour un utilisateur à un service déterminé en fonction de facteurs tels que le coût d'utilisation du service, les performances du réseau, les performances des hôtes et

l'état des files d'attentes des services.

- La fédération devra permettre la mise en oeuvre d'un système de facturation des coûts liés à l'utilisation des services. Cette facturation interviendra entre les différents membres de la fédération. Une ventilation pour chaque utilisateur devra être possible.
- L'utilisateur bénéficiera d'un traitement asynchrone de ses exécutions de services. Il pourra lancer des analyses et pendant ce temps réaliser d'autres tâches. Il pourra vérifier ponctuellement l'état d'avancement de ses exécutions de services ou attentes dans des files. Il pourra interrompre ses exécutions de services. Il pourra également récupérer les résultats de ses exécutions terminées quand il le jugera le moment opportun. Il pourra par exemple souscrire à un système d'envois de résultats par mail ou à un service de téléchargement de résultats sur un site FTP ou HTTP.

5.3 Cas d'utilisation

Cette section a pour but de présenter la synthèse des exigences de notre client (les différents membres de l'unité de bioinformatique de l'IBMM) sous forme de cas d'utilisation (Use Cases) UML.

5.3.1 Les différents types d'acteurs

Au cours de nos interviews et différents contacts avec les membres de l'IBMM, nous avons recensé quatre types de profils d'utilisateurs (normal, avancé, expert et bioinformaticien) du système ainsi qu'un profil technique (administrateur du système). Chacun des ces profils correspondra à un acteur de cas d'utilisation. Chaque acteur de profil supérieur hérite des cas d'utilisation du profil inférieur. A ces différents acteurs, nous ajouterons un acteur qui a un rôle mineur, le service comptable. Les caractéristiques des différents acteurs du système sont:

Normal: Cet acteur correspond au profil d'utilisation le plus basique. L'utilisateur normal du système désire utiliser des services bioinformatiques de la manière la plus transparente et la plus simple possible. Il ne désire pas se préoccuper de détails techniques et informatiques du choix du service à utiliser. Il ne veut se préoccuper que du choix d'un service d'un point de vue biologique. Ce profil correspond à un étudiant en biologie ou à un chercheur débutant qui a peu de connaissances des services bioinformatiques.

Avancé: Cet acteur correspond au profil d'utilisation de services bioinformatiques le plus courant. Le chercheur désire connaître et maîtriser un maximum de paramètres (informatiques et biologiques) lors de ses

utilisations de services bioinformatiques. Il désire également partager le fruit de ses recherches avec les autres membres de son organisation.

Expert: Cet acteur correspond à un profil d'expert. Il permet à l'utilisateur tout ce que le profil avancé permet mais à cela il ajoute la possibilité d'automatiser des traitements intermédiaires aux appels de services au moyen d'un langage de scripts.

Bioinformaticien: Cet acteur correspond au profil du bioinformaticien. Il s'agit d'un utilisateur possédant des compétences et l'expertise dans deux domaines, à savoir la biologie et l'informatique. Il doit avoir accès à tous les mécanismes de fonctionnement de la fédération. C'est lui le producteur des services bioinformatiques.

Administrateur système: C'est un acteur technique chargé de l'administration du système du point de vue de la gestion des ressources disponibles et de la sécurité (droits d'accès, politique de sécurité, etc.).

Service comptable: Cet acteur joue un rôle administratif. Il a pour seule vocation de facturer les services prestés à qui de droit.

5.3.2 Cas d'utilisation des utilisateurs du système

Cas d'utilisation globaux du système (figure 5.1)

- S'identifier auprès du système: Ce cas d'utilisation prend en compte la fonctionnalité qui permet à l'utilisateur de s'identifier auprès du système. Le système traitera ensuite les demandes de l'utilisateur en fonction de son identifiant d'utilisateur, de son profil ainsi que de son appartenance à différents groupes d'utilisateurs.
- Informations sur les quotas d'utilisation de ressources: Ce cas d'utilisation modélise la possibilité pour un utilisateur reconnu par le système d'obtenir des informations concernant sa consommation de ressources. Celles-ci lui sont attribuées par quotas.
- Obtenir la liste des services bioinformatiques disponibles: Ce cas d'utilisation prend en compte la fonctionnalité permettant à l'utilisateur d'obtenir la liste de tous les services disponibles dans le système. Seuls les services pour lesquels il a une permission d'utilisation lui seront proposés. Cette permission d'utilisation est liée à son identifiant d'utilisateur, à son profil ou aux groupes auxquels il appartient.
- Etat de l'exécution d'un service bioinformatique: Ce cas d'utilisation modélise la possibilité pour l'utilisateur de consulter l'état d'avancement de l'exécution d'un service qu'il aurait suscitée. Si l'exécution est terminée, l'utilisateur peut obtenir le résultat de l'exécution du service.
- Demande normale d'exécution d'un service bioinformatique: C'est l'adaptation du cas d'utilisation "Demande d'exécution d'un service bioinformatique" au profil de l'utilisateur normal.

- Demande avancée d'exécution d'un service bioinformatique: C'est l'adaptation du cas d'utilisation "Demande d'exécution d'un service bioinformatique" au profil de l'utilisateur avancé.
- Demande experte d'exécution d'un service bioinformatique: C'est l'adaptation du cas d'utilisation "Demande d'exécution d'un service bioinformatique" au profil de l'utilisateur expert.
- Demande d'exécution d'un service bioinformatique: Ce cas d'utilisation modélise l'utilisation complète d'un service pour lequel il est nécessaire de spécifier précisément tous les paramètres. Ce cas d'utilisation est la généralisation des 3 sous cas d'utilisation précédents.
- Gestion des services bioinformatiques personnels: (voir figure 5.2 et description des cas d'utilisation ci-après)

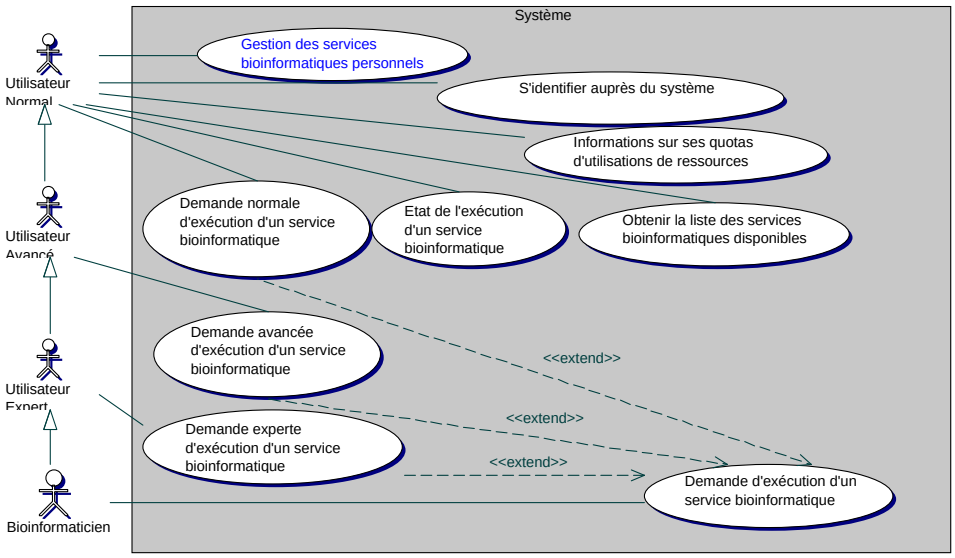


FIG. 5.1 – *Cas d'utilisation globaux du système*

Détails de la gestion des services bioinformatiques personnels (figure 5.2)

- Gestion normale des services bioinformatiques personnels: Ce cas d'utilisation est l'adaptation au profil normal du cas d'utilisation de la Gestion des services bioinformatiques personnels.
- Gestion avancée des services bioinformatiques personnels: Ce cas d'utilisation est l'adaptation au profil avancé du cas d'utilisation de la Gestion des services bioinformatiques personnels.

- Gestion experte des services bioinformatiques personnels: Ce cas d'utilisation est l'adaptation au profil expert du cas d'utilisation de la Gestion des services bioinformatiques personnels.
- Renommer un service bioinformatique personnel: Ce cas d'utilisation modélise la possibilité pour l'utilisateur de renommer le nom d'un de ses scripts de session d'utilisation. Il est important de différencier *session d'utilisation* et *session de connexion*. En effet, une *session d'utilisation* concerne la suite d'exécution d'un ou plusieurs services tandis qu'une *session de connexion* concerne la connexion de l'utilisateur au système. Au cours de cette *session de connexion*, il peut réaliser plusieurs *sessions d'utilisation* de services.
- Supprimer un service bioinformatique personnel: Ce cas d'utilisation illustre la possibilité pour l'utilisateur de supprimer un script de session d'utilisation qu'il aurait enregistré.
- Sauver un service bioinformatique personnel: Ce cas d'utilisation modélise la possibilité pour l'utilisateur de sauver le script de sa session d'utilisation. Ce script aura été généré automatiquement par le système. L'utilisateur aura ensuite la possibilité de "rejouer" ce script.
- Consulter les propriétés d'un service bioinformatique personnel: Ce cas d'utilisation illustre la possibilité pour l'utilisateur de consulter les informations concernant l'utilisation d'un service. Ce cas d'utilisation correspond à l'exigence de la nécessité d'un protocole complet joint au résultat de chaque utilisation de service.
- Changer les droits d'un service bioinformatique personnel: Ce cas d'utilisation modélise la possibilité pour un utilisateur de partager ou de ne pas partager un de ses scripts de session d'utilisation de services. Bien entendu, cette modification devra être avalisée par l'administrateur du système. Ce dernier sera le garant de la compatibilité de ce changement avec l'utilisation actuelle des ressources du système.
- Editer le script d'un service bioinformatique personnel: Ce cas d'utilisation modélise la possibilité pour l'utilisateur d'éditer un script de session d'utilisation . Au moyen de ces scripts, l'utilisateur pourra paramétrer finement ses appels aux services mais aussi crées des traitements automatiques entre des appels successifs de services ou encore créer des points de synchronisation entre d'éventuels appels de services qu'il aurait lancés en parallèle.

5.3.3 Cas d'utilisation d'administration du système

Cas d'utilisation d'administration du système (figure 5.3)

- S'identifier: Ce cas d'utilisation permet de modéliser la possibilité pour l'administrateur du système de s'identifier auprès du système.

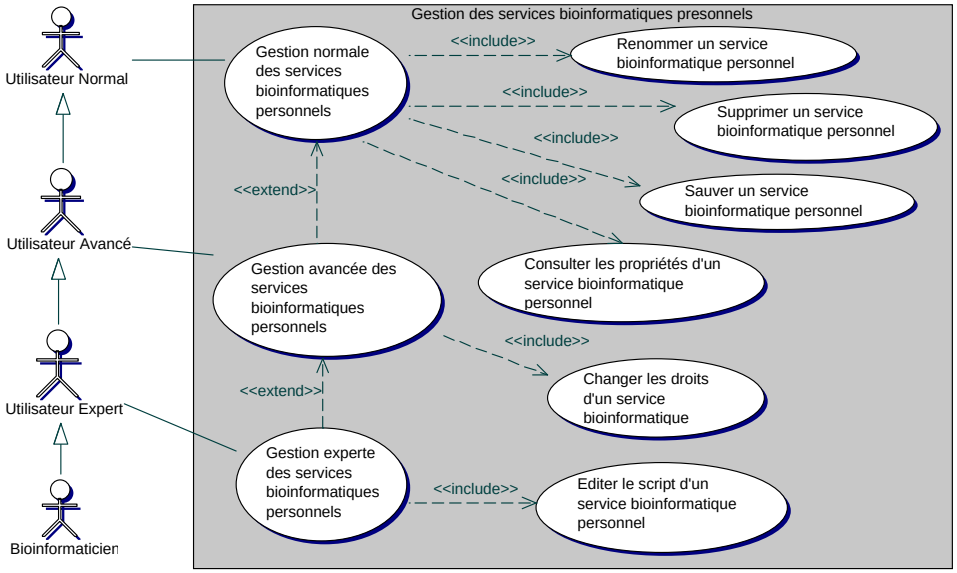


FIG. 5.2 – *Détails du cas d'utilisation de gestion des services bioinformatiques personnels*

- Gestion des logs et audit: Ce cas d'utilisation illustre la possibilité pour l'administrateur du système de consulter les événements qui se sont produits pendant la période d'activité du système.
- Gestion des utilisateurs: Ce cas d'utilisation factorise les différents cas d'utilisation de gestion des utilisateurs.
- Gestion des profils utilisateurs: Ce cas d'utilisation modélise la possibilité pour l'administrateur du système de paramétrer les différents profils (modification des membres).
- Gestion des groupes utilisateurs: Ce cas d'utilisation illustre la fonctionnalité offrant la possibilité pour l'administrateur du système de paramétrer les différents groupes (ajout/suppression/modification des membres).
- Ajout/Suppression d'utilisateurs: Ce cas d'utilisation modélise la possibilité pour l'administrateur du système d'ajouter ou de supprimer des utilisateurs.
- Gestion des droits des utilisateurs: Ce cas d'utilisation illustre la possibilité pour l'administrateur du système de modifier les droits d'un utilisateur particulier pour un service particulier.
- Gestion des services bioinformatiques: Ce cas d'utilisation factorise les différents cas d'utilisation de gestion des services bioinformatiques.
- Configuration des droits d'accès aux services bioinformatiques: Ce cas

d'utilisation modélise la possibilité pour l'administrateur de configurer les différents droits des utilisateurs, profils ou groupes pour un service déterminé.

- Ajout/suppression d'un service bioinformatique: Ce cas d'utilisation modélise la possibilité pour l'administrateur d'ajouter ou de supprimer un service bioinformatique et de l'inscrire auprès du système.
- Accepter un nouveau service bioinformatique: Ce cas d'utilisation modélise la possibilité pour l'administrateur d'accepter la demande d'inscription d'un service bioinformatique soumise par un administrateur de service bioinformatique.
- Gestion des ressources physiques: Ce cas d'utilisation modélise la possibilité pour l'administrateur du système de modifier la politique de gestion des ressources physiques. Il peut modifier l'assignation de composants à des hôtes, la réplication de composants, la politique de la gestion de la charge système, etc.
- Facturation des utilisations de services bioinformatiques: Ce cas d'utilisation illustre la possibilité pour le service comptable d'obtenir les informations de consommation de services bioinformatiques en vue de la facturation.
- Gestion des connexions à la fédération: Ce cas d'utilisation modélise la possibilité pour l'administrateur du système de gérer les connexions de son organisation aux membres de la fédération.
- Gestion des contrats entre membres de la fédération: Ce cas d'utilisation modélise la possibilité pour l'administrateur du système de sélectionner la manière dont les services bioinformatiques de son organisation seront diffusés auprès des autres membres de la fédération.
- Gestion du load-balancing: cas d'utilisation modélisant la possibilité pour l'administrateur du système de modifier les règles de load-balancing.

Cas d'utilisation d'administration d'un service bioinformatique (figure 5.4)

- Inscrire un service bioinformatique: Ce cas d'utilisation modélise la possibilité pour l'administrateur d'un service bioinformatique de l'inscrire auprès du système.
- Désinscrire un service bioinformatique: Ce cas d'utilisation modélise la possibilité pour l'administrateur d'un service bioinformatique de le désinscrire auprès du système.
- Modifier les informations d'état d'un service bioinformatique: Ce cas d'utilisation modélise la possibilité pour l'administrateur d'un service bioinformatique de modifier les informations le décrivant au sein du système.

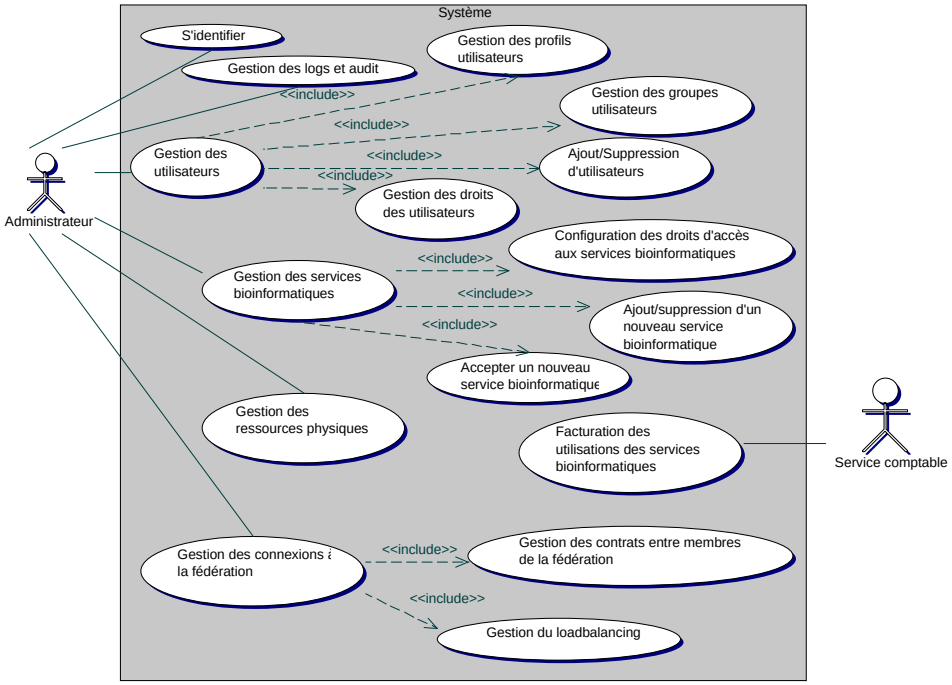


FIG. 5.3 – *Cas d'utilisation d'administration du système*

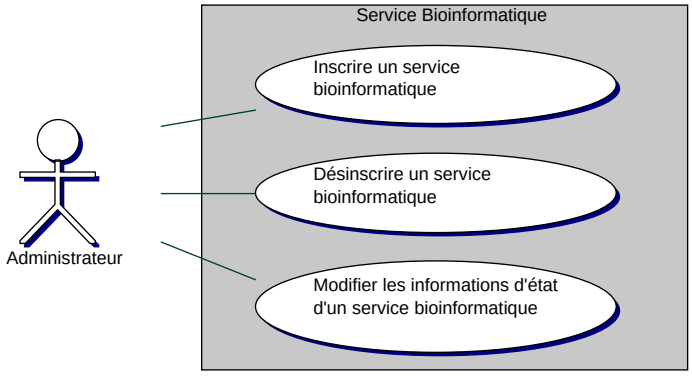


FIG. 5.4 – *Cas d'utilisation d'administration d'un service bioinformatique*

5.4 Diagramme de classes de haut niveau

Ce diagramme de classes de haut niveau (voir figure 5.5 page 74) permet de dégager, d'un point de vue conceptuel, les principaux concepts clés du domaine de l'analyse des exigences et leurs inter-relations pour permettre de mieux les comprendre. Le but de ce diagramme est purement didactique et ne représente aucunement des classes d'implémentation.

Un **Site** est caractérisé par un nom, une localisation physique (c'est-à-dire son adresse) et des informations pour contacter le responsable du site. Un site peut également faire partie intégrante d'un autre site. Nous avons dès lors une représentation hiérarchique des sites. (ex: L'Institut de Biologie et Médecine Moléculaire fait partie du Département de Biologie Moléculaire qui lui-même fait partie de l'Université Libre de Bruxelles). Un site se voit attacher plusieurs utilisateurs .

Un **Utilisateur** est caractérisé par un login, un mot de passe, un profil et des informations pour le contacter. Il est également attaché à un site. Un utilisateur peut être subordonné à un supérieur et/ou il peut être le supérieur d'un autre utilisateur (relation de supervision hiérarchique entre utilisateurs).

Un **Administrateur** est un **Utilisateur**. Il valide les politiques d'accès à une ressource qui peuvent être définies par lui-même ou par un utilisateur (changement de droit sur un service bioinformatique personnel ou l'ajout d'un nouveau service). Il définit également les politiques de gestion des ressources physiques disponibles.

Une **Ressource** est caractérisée par un nom et une description. Une ressource appartient à un utilisateur du système. L'ensemble des ressources est utilisé par l'ensemble des utilisateurs du système. Une ressource peut dépendre d'une autre ressource pour son utilisation (Une ressource logicielle dépend d'une ressource matérielle pour pouvoir s'exécuter). Une ressource est située sur un site.

Une **Ressource** est, soit un **Service Bioinformatique**, soit une **Ressource Physique**.

Un **Service Bioinformatique** est, soit un **Outil d'analyse**, soit une **Source de données**.

Un **Outil d'analyse** est caractérisé par un type d'analyse, une version, une description de ses paramètres d'entrée ainsi qu'une description de

son résultat.

Une **Ressource physique** est caractérisée par un état (disponible, indisponible) et par des caractéristiques la décrivant.

Une **Ressource physique** est, soit du **Temps de calcul** (CPU), soit de la **Mémoire physique** (Volatile ou non).

Un **Service Bioinformatique** dépend toujours au moins d'une ressource physique.

Une **Politique de gestion** est définie par un administrateur en fixant une règle de priorité d'utilisation sur une ressource physique pour un profil, un groupe d'utilisateurs ou un utilisateur.

Une **Politique d'accès** régule l'accès à une ressource pour un ou des utilisateur(s). Elle est caractérisée par un droit défini par un administrateur ou un utilisateur. Si cette politique d'accès est modifiée par un utilisateur, elle doit être validée par un administrateur.

Ainsi dans notre exemple où un biologiste moléculaire souhaiterait récupérer deux gènes particuliers présents dans le génome de deux espèces et stockés dans deux banques de génomes différentes et, ensuite, comparer les deux séquences au moyen d'un service d'alignement pour connaître la "similarité" entre les deux gènes; cette suite d'opérations pourrait être modélisée de la manière suivante: le biologiste est un **utilisateur** attaché à un **site** (l'IBMM) dépendant d'un autre **site** (l'ULB). Il utiliserait trois **ressources** de type **service bioinformatique**: un **outil d'analyse** (alignement) et deux **sources de données**. Ces **ressources** sont elles-mêmes constituées de **ressources physiques** de type **temps de calcul** et **mémoire**. L'utilisation de ces **ressources** est conditionnée par leurs **politiques d'accès** pour cet **utilisateur** et par les **politiques de gestion** des **ressources physiques** toutes deux définies par l'**administrateur** du **site** (IBMM).

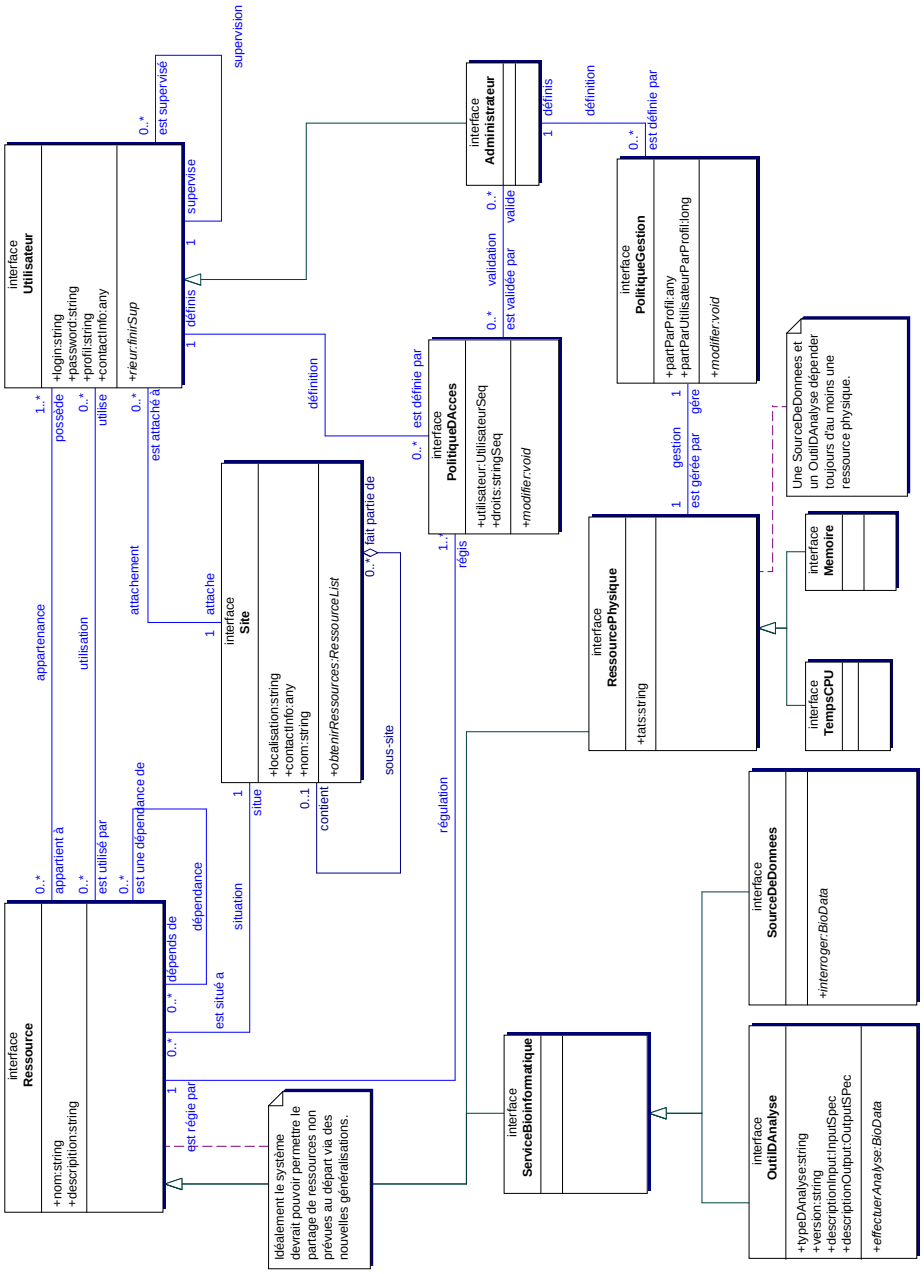


FIG. 5.5 – *Diagramme de classes de haut niveau*

5.5 Exigences non fonctionnelles

Sur base des différentes interviews, nous avons décelé les exigences non fonctionnelles suivantes :

- L'utilisation de la norme BSA de l'OMG qui est la seule norme objet définie et stable dans le domaine de la bioinformatique.
- L'utilisation de CORBA qui permet une implémentation plus aisée et rapide d'outils BSA.
- La capacité du système à migrer vers de nouvelles technologies de distribution et d'accès à des services (Services Web, J2EE, .NET, etc.).
- La possibilité d'utiliser dans le système de partage des outils plus anciens et non réimplémentables.
- L'utilisateur devra pouvoir rechercher et trouver un service bioinformatique de la manière la plus naturelle possible. L'outil de recherche de services le plus adéquat doit pouvoir répondre à la demande suivante : "Je veux réaliser un alignement entre les séquences A et B que je possède".
- Le système doit pouvoir garantir aux différents acteurs une utilisation sécurisée des services partagés au sein de la fédération.
- La nécessité de développer un modèle de description des services qui soit à la fois standardisé et parfaitement défini.
- Les services publiés par les différents membres de la fédération devront être accessibles par un maximum d'utilisateurs.
- Le système devra bénéficier d'une résistance à la charge maximale.
- Le système devra maximiser la "qualité de service" lors de l'utilisation de services.

5.6 Conclusion

Au travers des différentes interviews réalisées auprès des membres de l'IBMM nous avons isolé quatre classes d'utilisateurs selon leurs compétences en informatique et en bioinformatique. A ces utilisateurs s'ajoutent également les administrateurs des systèmes et services jouant un rôle technique (gestion de la sécurité, la maintenance, etc.) et un service comptable chargé de la facturation de l'utilisation des services bioinformatiques. Nous avons également isolé un certain nombre de cas d'utilisation (Use Cases) concernant ces différents acteurs.

Par la suite, nous avons réalisé un diagramme de classes modélisant de manière conceptuelle les éléments du domaine d'application, afin de formaliser notre compréhension et d'aider à sa validation. A ce diagramme s'ajoute bien sûr une série de définitions visant le même but.

Nous avons enfin relevé, sur base des interviews, les exigences non fonctionnelles de l'application telles que l'utilisation de la norme BSA et de CORBA ou l'ouverture de son architecture à d'autres technologies de distribution.

Les cas d'utilisation, la description du domaine d'application et la définition des exigences non fonctionnelles nous ont permis de valider notre vision de la problématique du futur système auprès des membres de l'unité de bioinformatique de l'IBMM. Ces éléments serviront de base au reste de notre analyse, principalement pour la phase de conception logique dans laquelle seront envisagées différentes solutions permettant de répondre à ces exigences.

Chapitre 6

Conception Logique

Le chapitre précédent nous a permis d'isoler les besoins fonctionnels et non fonctionnels émis par les différents acteurs du système. Certains de ces besoins peuvent être comblés de différentes façons. En effet, diverses solutions existent pour résoudre un même problème et celles-ci sont susceptibles d'influencer globalement, ou du moins en partie, l'architecture du système. Ce chapitre a pour but de présenter le cheminement que nous avons suivi pour aboutir à une solution répondant au mieux aux exigences que nous avons recensées.

6.1 Idées et principes de fonctionnement

Cette section a pour objectif de présenter certains besoins du système et les mécanismes que nous proposons pour y répondre.

6.1.1 Recherche de services

Présentation du besoin

La fédération offre à ses utilisateurs l'accès à une large collection de services bioinformatiques. Pour qu'un utilisateur puisse accéder à un service, à un moment ou à un autre, il doit être en mesure de le localiser, ou du moins, de récupérer les informations nécessaires à son utilisation. En outre, le système devant se suffire à lui-même, l'utilisateur ne doit pas chercher, en dehors du système, le service qu'il souhaite utiliser. En adéquation avec les exigences relevées, le système doit dès lors contenir un mécanisme de localisation des services permettant à ses utilisateurs de rechercher et de trouver les services correspondants à leurs besoins.

Pour y parvenir, la fédération doit être en mesure de construire un catalogue des services disponibles en son sein. Les médiateurs doivent, quant à eux, fournir une méthode d'accès à ce catalogue pour permettre aux utilisateurs de le consulter. Quels que soient les mécanismes de création du

catalogue et de consultation utilisés, le résultat reste similaire: une requête de service constituée d'une série de contraintes sur les propriétés du service est fournie par l'utilisateur. Une liste de services répondant à ces critères ainsi que les informations nécessaires à leur utilisation est retournée à l'utilisateur.

Une première solution assez simple serait de mettre en place un catalogue centralisé, accessible par tous les médiateurs de la fédération. Ceux-ci assureraient la mise à jour des informations concernant les services inscrits par leur biais et serviraient de lien entre les utilisateurs et le catalogue des services. Cette solution est envisageable dans la cas d'une fédération de taille restreinte. En effet, s'il existe peu de médiateurs partageant des services aux propriétés stables, les recherches, ajouts et modifications d'informations devraient ne pas être trop fréquents. Dans un tel cas, la charge imposée au catalogue resterait contrôlable. Mais si ce facteur de taille ou le nombre de mises à jour augmente, la charge imposée au catalogue risque rapidement de devenir trop importante.

Par ailleurs, pour répondre à l'exigence concernant le contrôle des accès et du partage de services au sein de la fédération, en vue d'une utilisation avec des partenaires privés, un mécanisme de limitation de la publication des services est nécessaire. Une solution reposant sur un catalogue centralisé nécessite une confiance absolue des fournisseurs de services envers le responsable du catalogue. Nous ne pensons pas qu'une telle confiance puisse être acquise par un partenaire unique au sein de la fédération. Sans cette confiance, le déploiement du système entre plusieurs partenaires pourrait être compromis. Dans ce but de tolérance à la montée en charge et de contrôle individuel des accès, une solution distribuée semble être la plus pertinente à envisager.

Traditionnellement, en informatique distribuée, lorsque la charge, qu'un composant d'un système risque de subir, est importante, la solution la plus couramment pratiquée est de le répliquer afin de répartir sa charge entre plusieurs hôtes du système. La fédération que nous souhaitons mettre en place ne déroge pas à cette règle.

Une autre solution, probablement moins immédiate, consisterait à ne pas créer, à proprement parler, de catalogue. Plutôt que de maintenir à jour, sous une forme quelconque, une liste des services existants, on utiliserait un mécanisme de recherche distribuée. A chaque fois qu'un utilisateur effectue une recherche de services, ce mécanisme assurerait que tous les médiateurs susceptibles de fournir les informations sur un service correspondant soient interrogés. Des modes de réalisation de ces deux types de solutions ont été envisagés, ils seront présentés avant que ne soit discuté le choix de l'un ou l'autre.

Réplication du catalogue

Parmi l'ensemble des médiateurs de la fédération, on peut considérer le catalogue comme une seule et même ressource externe, même s'il est distribué ou comme un composant intégré aux médiateurs. Dans le premier cas, la distribution du catalogue est indépendante des médiateurs. Dans le second cas chaque médiateur possède son propre catalogue. Le mécanisme de synchronisation des catalogues est alors intégré aux autres mécanismes inter-médiateurs.

Dans le cas d'une distribution du catalogue indépendante des médiateurs, le catalogue est considéré comme une ressource unique, externe aux médiateurs. Lorsqu'un médiateur désire consulter le catalogue il effectue un appel vers un composant qu'il croit unique sans savoir qu'il s'agit d'un composant distribué. Pour une telle distribution du catalogue, on peut envisager d'utiliser directement les mécanismes de replication et de distribution d'un SGBD [1]. Ces mécanismes pourraient peut-être limiter la complexité du catalogue des services, mais sa structure nous paraissant, a priori, peu complexe, ils restent envisageables. La relative simplicité de telles solutions les rend attrayantes.

Cependant, cette gestion indépendante du catalogue ne permet pas, ou difficilement, une gestion de la sécurité des accès aux informations concernant les services. Tous les services disponibles dans la fédération retrouveraient leurs informations au sein de tous les catalogues de la fédération, et ceci quels que soient les droits d'accès leur étant associés. Un service réservé à un ou deux médiateurs verrait ainsi son information distribuée dans tout le système.

Le choix d'une solution existante pour la gestion de la distribution du catalogue des services pose également un autre problème par rapport aux objectifs du système. En effet, l'architecture que nous proposons se veut ouverte et utilisable dans différents milieux avec différents outils informatiques hétérogènes. Ainsi, le choix de l'une ou l'autre solution existante risque de poser des problèmes de compatibilité devant soigneusement être pris en compte. Le choix d'une solution devrait se limiter dès lors aux systèmes de distribution bénéficiant de spécifications ouvertes et librement implémentables afin d'en assurer une disponibilité totale quelle que soit la plateforme logicielle ou matérielle.

De plus, les besoins du système de réplication du catalogue sont plus simples que ce que propose la plupart des outils de distribution génériques, seule une partie de leurs fonctionnalités s'avère requise. Les services bioinformatiques de la fédération ne s'inscrivent qu'auprès d'un seul médiateur. Lorsqu'un service est inscrit auprès d'un médiateur, seul celui-ci modifie ou ajoute de l'information concernant le service, les autres médiateurs se

1. Des SGBD tels que Oracle9i Database, Microsoft SQL Server 2000 ou PostgresSQL offrent ce genre de fonctionnalités.

limitent à la lecture de cette information. On considère également que l'information sur les services est relativement stable dans le temps, les services ne changeant pas leurs propriétés très souvent.

Par ailleurs, si l'on considère aussi que l'information concernant les services se trouve filtrée selon les droits d'accès qui lui sont associés et que seuls les médiateurs ayant accès à un service ont accès à ses informations, alors, plutôt que de construire un catalogue unique répliqué, il est également possible de maintenir un catalogue par médiateur. De la sorte, chaque médiateur est responsable de son propre catalogue, lequel ne contiendrait que les informations concernant les services auxquels il fournit un accès. Evidemment, le principe de fédération de partage implique que le catalogue d'un médiateur doit contenir plus que les seuls services inscrits auprès de celui-ci; une certaine réplication des données reste donc nécessaire.

Les médiateurs sont, dans ce cas de figure, également responsables de la propagation des informations concernant les services. Ainsi, lorsqu'un médiateur reçoit un ajout ou une modification d'informations, il en vérifie la validité pour ensuite faire suivre l'information aux médiateurs auxquels il est directement connecté. Un ajout d'information n'est valide que s'il concerne un service pour lequel aucune information n'est disponible. Une modification n'est valide que si elle est postérieure à l'information déjà contenue dans le catalogue. On évite ainsi les bouclages d'information entre les médiateurs. Si un médiateur a déjà vu passer un ajout ou une modification, il ou elle sera obligatoirement considérée comme non valide et ne sera plus propagée.

Pour répliquer l'information, seuls trois types de messages sont nécessaires: un message d'ajout lorsqu'un service s'inscrit au sein de la fédération, un message de modification lorsque les propriétés d'un service sont mises à jour et un message de suppression lorsque le service se désinscrit de la fédération. Pour vérifier la validité d'une mise à jour, on utilise un numéro de version de l'information incrémenté à chaque modification. Afin de maintenir la cohérence des informations dans la fédération, seul le médiateur auquel est attaché un service peut générer un message de suppression de service ou un message de modification de propriétés doté d'un nouveau numéro de version.

Lorsque toutes les connexions entre médiateurs sont déjà réalisées, ce mécanisme assure la mise à jour de tous les catalogues de la fédération. Cependant, la fédération est un réseau dynamique; des médiateurs peuvent la quitter ou la rejoindre à tout moment. Lorsqu'un médiateur quitte la fédération, il doit s'assurer que tous ses services sont également retirés du catalogue. Lorsqu'ils se déconnectent de la fédération, les médiateurs doivent émettre des messages de suppression pour chacun de leurs services.

Quand un médiateur rejoint la fédération, il ne peut pas se contenter des messages postérieurs à son arrivée pour construire son catalogue. Il doit dès lors demander à ses voisins déjà présents de lui envoyer le contenu de leur propre catalogue. Le processus est, cependant, plus complexe. Lorsque deux

médiateurs se connectent, ils doivent s'assurer que leur catalogue respectif contient bien les mêmes informations. Pour y parvenir, ils doivent tous deux s'échanger la totalité de leur catalogue. L'échange de catalogue peut se faire de manière simple, il suffit que les deux médiateurs s'envoient mutuellement des messages d'ajouts pour tous leurs services respectifs, l'un découvrant ainsi ceux de l'autre.

Un message d'ajout est invalide lorsqu'il porte sur un service déjà inscrit en catalogue. Si les médiateurs qui se connectent, s'envoient de l'information sur des services qu'ils connaissent déjà tous les deux, elle sera considérée comme invalide et mise de côté. Si la fédération se trouve dans un état cohérent et si tous les médiateurs connaissent un service, alors ils connaissent la même *version* de l'information qui y est attachée.

Pour éviter que le catalogue de la fédération ne contienne des informations sur des services qui ne seraient plus partagés, pour une raison ou pour une autre[2], un médiateur peut supprimer un service qui ne lui est pas directement inscrit. Ainsi, lorsqu'un médiateur tente d'accéder à un service, si celui-ci ne peut pas être contacté, c'est qu'il n'est plus partagé au sein de la fédération. Dans ce cas, le médiateur qui a tenté l'accès peut envoyer un message de suppression, en considérant qu'il s'agit d'un *oubli* de la part du médiateur auprès duquel était inscrit le service. Ce message est traité comme tous les autres messages de suppression. Dans le cas où un médiateur reçoit un message de suppression pour un service qui lui est inscrit, il traite, dans un premier temps, le message normalement et dans un second temps il réémet un message d'inscription de ce service auprès de l'ensemble de la fédération.

Le système de réplication tel que décrit ci-dessus, fournit tous les mécanismes nécessaires à la gestion du catalogue des services. Toutefois, nous avons envisagé d'autres mécanismes pour mettre au point ce catalogue. Ces systèmes ne se basent plus sur la constitution d'un catalogue répliqué mais sur une forme plus complexe de distribution du catalogue et de recherche.

Recherches distribuées

Plutôt que de construire un ou des catalogues de l'ensemble des services disponibles consulté(s) en une seule fois, il est possible d'utiliser des mécanismes se basant sur une distribution de la recherche. Ces méthodes sont souvent utilisées dans les systèmes décentralisés de type P2P (voir section 2.7.1 page 40). Il s'agit en fait de méthodes d'*exploration* de la fédération vue comme un réseau de noeuds, chaque noeud étant un médiateur. Un message de requête *explore* le réseau de médiateurs à la recherche du ou des médiateurs qui détiendraient une réponse. Lorsqu'une réponse est trouvée, elle suit alors le chemin inverse de la recherche pour retourner à son émetteur.

2. Par exemple, lorsqu'un médiateur a été brutalement arrêté et qu'il n'a pas pu désinscrire ses services.

Il existe plusieurs systèmes de *parcours* du réseau formés par les médiateurs. On distingue généralement deux types d'approches, l'une où aucune structuration de l'information sur les services n'est mise en place et l'autre concernant les méthodes avec structuration de l'information. Si aucune structuration de l'information n'est effectuée, il n'y a aucune distribution de l'information sur les services entre les médiateurs du système, seuls les messages de recherche sont distribués. Avec une structuration de l'information, l'ensemble du réseau est vu comme un seul catalogue dont les membres ne contiennent qu'une partie de l'information totale. L'information concernant les services est distribuée entre les différents médiateurs selon un mécanisme précis. Une recherche est réalisée en contactant le ou les médiateurs susceptibles de contenir l'information.

Dans un réseau sans structuration de l'information, pour assurer qu'une recherche permet de trouver tous les services correspondants à ses critères, il est nécessaire que tous les médiateurs de la fédération soient interrogés. Si un médiateur n'est pas interrogé, et qu'il est le seul à connaître ses services, l'émetteur de la recherche ne pourra jamais être sûr que tous les services disponibles, correspondants à ses critères de recherche, lui ont été retournés. Pour y parvenir on peut utiliser un système, dit de *query flooding*[3] dans lequel les recherches de services sont transmises de médiateurs en médiateurs au travers de toute la fédération (figure 6.1).

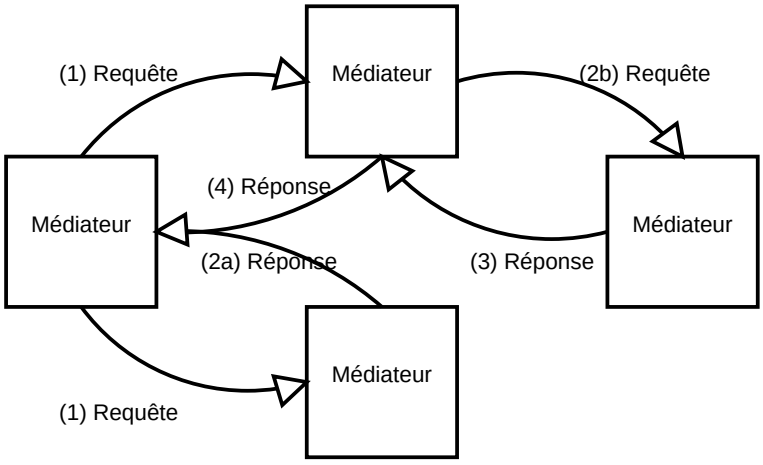


FIG. 6.1 – *Messages échangés suivant le principe du* Query Flooding

Lorsqu'un médiateur effectue une recherche, il envoie à tous ses voisins une requête de service contenant les contraintes de sa recherche (1 sur la figure 6.1). A chaque fois qu'un médiateur reçoit une requête de service, il vérifie s'il peut y répondre. Si oui il envoie une réponse au médiateur

3. Les recherches sont propagées entre tous les membres, les *requêtes inondent* le réseau des participants au système.

d'où provient la requête (2a & 3). Il retransmet également la requête à ses autres voisins (2b). Pour empêcher les boucles, un médiateur ne doit pas traiter plusieurs fois la même requête. A cette fin, chaque requête doit se voir attribuer un identifiant unique.

Quand il voit passer une réponse, un médiateur la transmet au médiateur dont il a reçu la recherche correspondante (4). Pour permettre cet acheminement des réponses vers les émetteurs des recherches correspondantes, chaque médiateur doit maintenir une *table de routage* des requêtes. Comme pour les recherches, un médiateur ne doit pas traiter plusieurs fois la même réponse. Un identifiant unique doit aussi être attribué aux réponses.

Ce principe, simple en apparence, n'est ni aisé à implémenter dans le cadre d'une couverture totale du réseau [4], ni adapté à un réseau de taille importante. Sur un grand réseau, chaque requête nécessite autant de messages que de médiateurs présents au sein de la fédération ce qui entraîne que la charge de chaque noeud croît plus vite que la taille du réseau. Le maintien permanent de toutes les entrées des *tables de routage* pour l'acheminement des réponses est également impossible. Il en résulte des risques de pertes de réponses. Le temps de parcours de tous les noeuds est indéfini et probablement long, l'émetteur d'une requête ne sait donc jamais si une réponse peut encore lui arriver. Le système ne peut pas garantir la découverte de tous les services correspondants à une requête; des réponses peuvent se perdre et d'autres arriver alors que l'émetteur de la requête ne les attend plus.

Lorsque les recherches ne nécessitent pas de couverture totale du réseau, diverses améliorations peuvent être envisagées comme un temps de vie limité pour les messages de requêtes et de réponses. Ceci permet de limiter la charge du système ou une limitation de la portée des recherches à un sous-ensemble des médiateurs de la fédération. Des mécanismes de *caches* plus ou moins complexes permettent également d'alléger la charge en terme de messages et de diminuer le temps de réponse. Cependant, ils ajoutent une latence dans la mise à jour de l'information. Dans un système avec *caches* avec une latence de mise à jour importante, il se peut que des services sortis de la fédération soient toujours retournés en réponse à des requêtes car ils sont toujours inscrits dans les *caches* de certains médiateurs. De même, les services nouvellement inscrits peuvent rester ignorés parce qu'ils ne sont pas encore inscrits dans toutes les *caches*.

Les solutions plus complexes, basées sur une structuration et une distribution de l'information permettent de palier au problème de la montée en charge tout en conservant une totale couverture de recherche. Ces solutions organisent l'information concernant les services disponibles au sein de la fédération selon une structure de données partagée entre tous les médiateurs. Il existe plusieurs solutions spécifiées se basant sur des tables de hachage dis-

4. Dans un système en *query flooding*, lorsque tous les participants sont assurés de recevoir toutes les requêtes, on parle de *couverture totale du réseau*.

tribuées ou *Distributed Hash Table*([RD01a][SMK+01]). Dans les systèmes *DHT*, pour chaque service, une clef d'accès à l'information le concernant est calculée et sert à la répartition de cette information entre les noeuds. Chaque noeud se voit confier le maintien des entrées de la table couverte par un certain intervalle de clefs. Lorsqu'un noeud est responsable d'une clef, il connaît les informations du service qui lui est associé.

Comme les informations sont stockées par clef, les recherches se basent sur l'existence de celles-ci et une requête correspond à la jointure des résultats de plusieurs interrogations de la *DHT*. Comme pour le *query flooding*, le mécanisme d'interrogation de la *DHT* se base aussi sur une transmission de messages entre les médiateurs. Mais ici, les connexions entre médiateurs sont structurées de telle sorte qu'un médiateur peut toujours décider lequel de ses voisins est le plus proche du médiateur qui maintient l'information pour une clef donnée. Le message d'interrogation de la clef est ainsi transmis de médiateurs en médiateurs via le meilleur chemin de manière à atteindre le médiateur qui détient l'information associée à la clef.

Le principale défaut des systèmes *DHT* est la nécessité du stockage par clef. Une recherche doit se baser sur l'extraction d'information par association de clef. Des requêtes complexes peuvent être envisagées via la création de multiples clefs par service mais cela augmente alors la charge en terme de messages transmis dans le système et complexifie la gestion des résultats des interrogations. Dans la cadre de notre fédération, les connexions entre médiateurs sont effectuées de manière arbitraire par leurs administrateurs alors que le système de *DHT* repose sur une structuration de ces connexions.

Choix d'une solution

Dans le cadre de la fédération de partage de services bioinformatiques hétérogènes, le système le plus souhaitable doit fournir un mécanisme de recherche couvrant la totalité des services disponibles avec une montée en charge qui resterait gérable. Ces exigences empêchent l'utilisation de mécanismes à base de *query flooding*, puisqu'ils consomment trop de ressources ou qu'ils limitent la portée des recherches à un sous-ensemble des services disponibles. Les différentes améliorations du système rencontrées ne permettent pas de palier à ces défauts. C'est pour cette raison que nous n'avons pas retenu ce type de mécanisme.

Les mécanismes à base de *DHT* semblent promettre beaucoup de résultats pour de nombreuses applications ([CDKR02][RD01b]). Néanmoins, notre système doit permettre des recherches de services basées sur des critères complexes et de multiples contraintes. Cependant, les mécanismes à base de *DHT* délivrent toute leur puissance lors de requêtes simples et nécessitent une structuration des connexions entre médiateurs par le système lui-même, ce qui n'est pas compatible avec les exigences relevées.

Nous avons dès lors choisi d'utiliser le mécanisme de réplication du ca-

talogue au sein de chaque médiateur. Dans le cas présent, le système est prévu pour travailler avec des médiateurs et des services stables. La charge imposée par la réplication des données devrait dès lors rester gérable.

6.1.2 Techniques d'appels aux services

Présentation du besoin

Une fois que l'utilisateur a trouvé le service correspondant à ses besoins, un mécanisme pour y accéder est nécessaire. Ce mécanisme doit permettre à un utilisateur d'accéder à un service qui soit local au médiateur ou distant tout en respectant les droits d'accès sur ce service. Cet accès, en respectant ces conditions, doit être transparent pour l'utilisateur.

Les *chaînes de proxies* proposées dans [DD03]

Dans la fédération, les services sont des objets correspondants aux interfaces définies par la norme BSA(voir section 1.4 page 9). Le mécanisme d'accès à ces services doit permettre aux utilisateurs de services partagés au sein de la fédération d'y accéder en tant qu'objet BSA. Pour y parvenir, dans un premier temps, l'utilisateur avertit son médiateur qu'il veut utiliser un service et reçoit en retour les objets BSA nécessaires pour l'interaction avec le service désiré.

Comme définis dans [DD03], les objets manipulés par l'utilisateur sont des *proxies*, ou composants façades vers le service. Un *proxy* est une *image* du service réel située au sein d'un médiateur. Lorsque l'utilisateur souhaite accéder à un service, il communique avec un *proxy* situé au sein de son médiateur. Un mécanisme d'indirection des communications permet de faire transiter les messages entre le *proxy* avec lequel communique l'utilisateur et le service lui-même. L'indirection des communications entre le *proxy* et le service se fait au moyen du mécanisme des *chaînes de proxies* (figure 6.2) tel proposé dans [DD03].

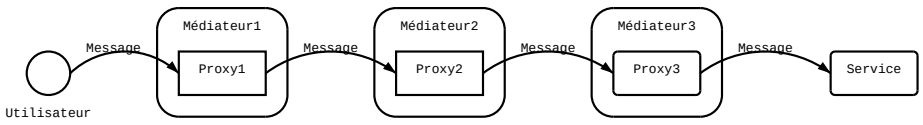


FIG. 6.2 – *Indirection de la communication au moyen d'une* chaîne de proxies

Ce système de *chaînage des proxies* permet de masquer complètement la localisation physique des éléments de la fédération à l'utilisateur. Les mêmes mécanismes sont utilisés pour accéder à des services locaux du médiateur ou à des services situés auprès d'un médiateur distant. Il permet aux applications utilisant la fédération de travailler avec les services distants comme s'il

s'agissait de services directement accessibles. Le fonctionnement et la structure de la fédération n'a pas d'influence sur l'utilisation des services par les éventuelles applications.

Dans [DD03], la *chaîne de proxies* pour accéder à un service depuis un médiateur donné est créée de manière statique en même temps que l'inscription du service auprès de son médiateur. Ce système impose de mettre à jour les *proxies* au sein de chaque médiateur si une modification de la fédération a lieu. De plus, si cette *chaîne* est présente à l'avance, elle ne peut pas tenir compte des caractéristiques dynamiques de la fédération telles que l'absence temporaire ou la surcharge d'un médiateur ou d'un service.

C'est pourquoi nous proposons que la *chaîne de proxies* nécessaire pour accéder à un service depuis un médiateur donné ne soit créée qu'au moment où un utilisateur de ce médiateur accède au service. Une *chaîne de proxies* devient donc à usage unique, un seul utilisateur l'utilise pour une seule utilisation du service. Même si plusieurs utilisateurs d'un même médiateur accèdent à un même service, ils utiliseront chacun une *chaîne* différente. Pour créer la *chaîne de proxies*, on utilise le mécanisme de *routes* présenté ci-dessous.

Des *chaînes de proxies* aux *routes*

Pour chaque appel, les médiateurs sélectionnent, au moment de l'accès, la meilleure *chaîne* possible pour atteindre le service demandé. Nous appellerons ce principe de création dynamique de *chaînes de proxies*, **la création de routes**.

Ce système de création de *routes* est dynamique et permet un indéterminisme concernant le service réellement utilisé. Ainsi, si un utilisateur désire utiliser un service de type A sans se soucier de sa localisation, un système dynamique peut effectuer, de manière invisible pour l'utilisateur, un choix parmi plusieurs services de type A disponibles. On peut donc obtenir un système de sélection du *meilleur service* selon des critères définis au moment de l'accès. Une partie de ces critères peut même être laissée au choix de l'utilisateur. Libre à lui, par exemple, de choisir entre un service lent mais gratuit ou un service rapide mais coûteux.

Cependant, il persiste un autre problème inhérent à l'utilisation de la norme BSA dans les *chaînes de proxies* ou les *routes*. En effet, le *chaînage des proxies* repose sur la retransmission des messages de médiateurs en médiateurs. Ces messages sont dupliqués entre chaque médiateur. Le service et le client travaillent toujours avec des objets locaux (situés au sein de leur médiateur). Par ailleurs, la norme BSA prévoit l'utilisation des services avec des arguments de n'importe quel type. Ceci inclut des arguments pouvant être des structures complexes ou des objets partagés entre plusieurs applications. La duplication des messages empêche dès lors une réelle utilisation de cette fonctionnalité, puisque les arguments se voient copier de *proxy* en

proxy sans qu'il n'y ait une réelle synchronisation entre les différentes copies. Supposons un service qui modifie et insère son résultat dans un objet passé en paramètre (figure 6.3). Si le client passe un objet en argument à son *proxy* (`2. appel(UnObjetParam)`), celui-ci sera copié jusqu'à un service censé modifier l'objet (`2.1.1. create`). Une fois le traitement terminé, seule la copie de l'objet situé au sein du médiateur du service a été modifiée (`2.1.2. ecrireRésultat`). Si le client tente de récupérer le résultat dans l'objet localisé au sein de son médiateur (`3. lireRésultat`) il n'accédera pas à l'objet qui a été modifié.

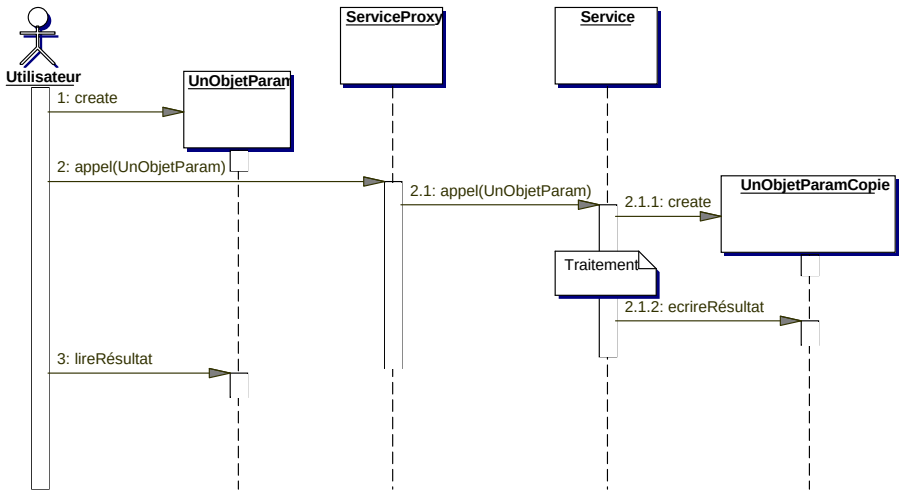


FIG. 6.3 – *Diagramme de séquence illustrant que l'objet donné en paramètre n'est pas modifié par le service.*

Pour palier à ce problème, il faudrait mettre en place un système capable de gérer la création de *proxies*, non seulement pour les services eux-mêmes mais également pour les structures ou objets passés en arguments (figure 6.4. Cependant, il ne s'agit pas d'une solution évidente à implémenter. De plus, comme la section `BioObjects` de la norme BSA ne prévoit pas d'arguments de ce type, nous avons choisi de ne pas conserver cette solution. La solution proposée permet uniquement l'utilisation de services recevant des paramètres qui ne sont plus modifiés une fois l'accès initié. De même, après traitement, le résultat d'un service ne peut plus être modifié une fois qu'il a été transmis à l'utilisateur.

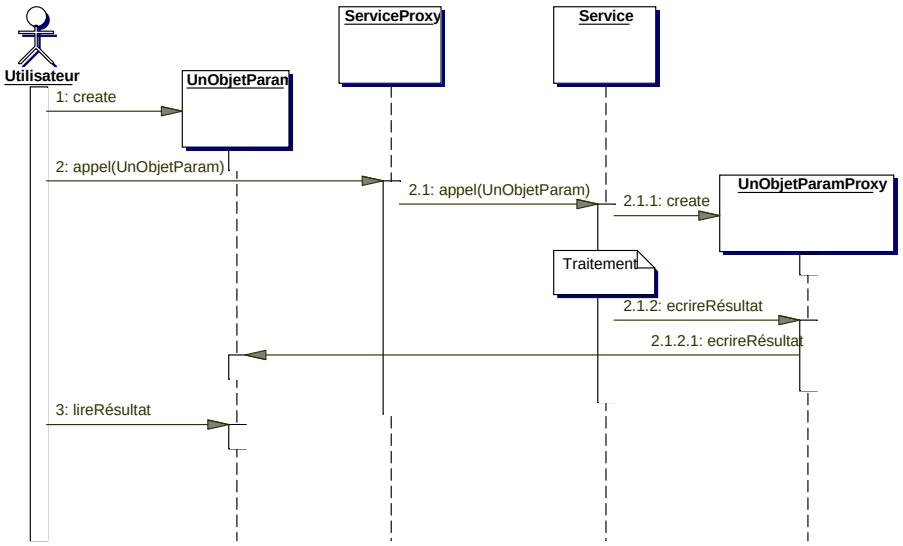


FIG. 6.4 – *Diagramme de séquence illustrant la création de* proxies *de paramètres.*

Les *routes* et la *qualité de service*

L'utilisation de la norme BSA pour décrire un service bioinformatique impose le choix de son identifiant. Ainsi, un service bioinformatique est identifié par les champs `Type`, `Name`, `Supplier`, `Version`, `Installation` et `Description` présents dans la spécification de l'`AnalysisType` de la norme BSA.

Cependant, il ne faut pas négliger qu'au sein de la fédération, plusieurs services peuvent correspondre à un même `AnalysisType`. En effet, la possibilité offerte par le mécanisme de *création de routes* peut entraîner qu'un même service soit accessible par des *routes* différentes. Néanmoins, ces *routes* peuvent avoir des caractéristiques différentes. De plus, c'est sur base de ces caractéristiques qu'un utilisateur de service bioinformatique, opérera son choix si deux services offrent la même analyse. Par exemple, s'il a le choix entre un service gratuit ou un service payant, il choisira certainement le service gratuit. Bien entendu, ces caractéristiques, autres que le type d'analyse sont à définir. Nous appellerons ces caractéristiques, les *critères dynamiques d'évaluation d'une route.*

Les différents critères dynamiques d'évaluation d'une *route*

Tout d'abord, il est important de remarquer que deux `AnalysisType` seront considérés comme équivalents si les champs `Type` et `Version` de leur `AnalysisType` BSA sont identiques. Les autres critères de description d'un

service seront les autres champs de l'`AnalysisType` que nous appellerons *critères statiques* et les *critères dynamiques* correspondants à la *qualité de service* liée à l'exécution du service bioinformatique.

Ces critères dynamiques doivent prendre en considération la charge du système en terme de temps CPU monopolisé, du nombre threads déjà en exécution, de la mémoire occupée, du nombre de places occupées dans les éventuelles files d'attente ainsi que d'éventuels coûts liés à l'exécution du service. Ces critères dynamiques permettront le choix du meilleur service en terme de *qualité de service* parmi des services identiques d'un point de vue "biologique". Ainsi pour effectuer un appel, le système se chargera automatiquement de choisir le meilleur service, en respectant la politique de gestion des ressources et des coûts du médiateur, ainsi que les contraintes en terme de *qualité de service* fixées par l'utilisateur.

Il apparaît dès lors que le choix de ces critères dynamiques doit être le plus pertinent possible, en adéquation avec les attentes des futurs utilisateurs du système.

Nous avons donc dressé une liste de critères qui nous semblait a priori la plus pertinente pour l'évaluation de la *qualité de service* d'un service bioinformatique:

- La charge du CPU de la machine hôte du service. Cette charge doit prendre en compte le nombre de processus système en cours, de threads au sein d'applications en exécution mais aussi éventuellement la charge de calcul répartie sur un cluster d'hôtes.
- La charge au niveau mémoire physique. Il peut s'agir de la mémoire volatile (RAM) ou de la mémoire de masse qui peuvent influencer le choix d'un ou l'autre service.
- La place dans une éventuelle file d'attente. Dans les services existants, les demandes d'analyses sont exécutées une à la fois, les demandes sont stockées dans une file "first in first out" (fifo) à laquelle est souvent ajouté un mécanisme de priorité. Certains utilisateurs sont prioritaires sur les autres. On retrouve en général, trois niveaux de priorité.
- Le temps d'exécution moyen du service: le temps moyen de traitement des dix (par exemple) dernières exécutions.
- Le temps de la dernière exécution du service.
- Le temps d'exécution de l'analyse en cours.
- Le prix de l'utilisation du service.

Sur base de cette liste de critères dynamiques d'évaluation, nous avons opéré un choix. En effet, il n'est pas opportun de les utiliser tous, certains se prêtent mieux à l'évaluation par un utilisateur tandis que d'autres non.

C'est pour cette raison que nous avons décidé de ne pas conserver les paramètres de charge CPU et de charge mémoire car ils ne sont pas très explicites pour un utilisateur et ne permettent pas le choix du service offrant la combinaison coût et temps d'exécution la plus favorable. Par contre, ces

critères peuvent être très utiles pour évaluer une éventuelle répartition de la charge CPU et/ou de la charge mémoire ainsi que pour déterminer la nécessité de répliquer un composant.

Nous avons dès lors retenu les critères suivants:

- La place dans une éventuelle file d'attente.
- Le temps d'exécution moyen du service.
- Le temps de la dernière exécution du service.
- Le temps d'exécution de l'analyse en cours.
- Le prix de l'utilisation du service.

Principes de fonctionnement des *routes*

Une fois la recherche dans le catalogue et le choix d'un service effectués par l'utilisateur, ce dernier peut enfin réaliser son appel au service en fournissant les arguments d'entrée et les critères de qualité de service les plus importants pour lui. Dès cet instant, le médiateur réalise une recherche de *routes* candidates à travers la fédération. Une fois l'ensemble des réponses collectées, le système choisira la *route* constituant le meilleur équilibre entre les contraintes fixées par l'utilisateur et celles de la politique du choix de service du médiateur. Le choix opéré, le médiateur va déclencher un mécanisme de *création de route* et ensuite procéder à l'appel du service.

Une fois l'appel terminé, le médiateur détruit la *route* et conserve le résultat de l'exécution du service afin que l'utilisateur puisse le récupérer par après.

Il est important de remarquer que, si une *route* n'est plus disponible entre la phase de recherche et celle de création, le médiateur choisira une autre *route* considérée comme *équivalente*. Ce choix peut être effectué par chaque médiateur présent sur la *route*. De même, s'il n'existe plus de *route* pour atteindre un service, celui-ci sera désinscrit.

Nous avons préféré ce mécanisme à celui d'une réservation de *routes*. Au moyen d'un tel mécanisme, la recherche aurait pour but de réserver toutes les *routes* candidates avant le choix de la *meilleure* par le médiateur. Ce système aurait pour désavantage d'imposer la gestion de la *déréservation* des *routes*. Ceci, il faut le reconnaître, est gourmand en ressources (Création des *routes de proxies* et destruction de ces *routes*).

6.1.3 Sécurité

Présentation du besoin

En parcourant les exigences fonctionnelles et non fonctionnelles du système, on peut aisément remarquer que la sécurité et le contrôle des accès sont des éléments essentiels qui conditionneront l'utilisation du système en

situation réelle. En effet, une organisation utilisant le système doit pouvoir réguler l'accès aux différents services pour ses propres utilisateurs mais aussi pour les autres membres de la fédération. Ainsi, un organisme public de recherche doit être à même de contrôler l'accès de ses chercheurs à ses ressources éventuellement payantes. De plus, il doit pouvoir garantir aussi à un partenaire privé, avec lequel il aurait des accords d'échange d'utilisation de services, que les services échangés ne sont pas utilisés par d'autres membres de la fédération.

Pour ces raisons nous proposons une gestion de la sécurité et des accès à deux niveaux:

- Au niveau intra-médiateur
- Au niveau inter-médiateurs

La sécurité intra-médiateur

La sécurité intra-médiateur concerne les relations entre les utilisateurs (clients) et le médiateur.

Un utilisateur doit être identifié auprès d'un médiateur mais doit rester inconnu des autres membres de la fédération. En effet, il n'y a pas de contrôle central des accès pour tous les utilisateurs des différents médiateurs de la fédération. Ainsi, chaque médiateur a pour mission d'assurer, pour ses propres utilisateurs, la vérification de l'utilisation des services qu'il connaît. Ainsi, un médiateur définit les services de son catalogue auxquels ses utilisateurs auront accès.

L'accès aux services bioinformatiques est régi par un système de droits. Ces droits d'accès sont définis de manière à être attribués à un utilisateur, à des profils d'utilisateurs ou à des groupes d'utilisateurs. Ces groupes et profils ont pour objectif de simplifier et de faciliter la tâche de l'administrateur du système.

En vue de réaliser l'objectif d'identification de l'utilisateur auprès du système, nous proposons le système suivant: l'utilisateur s'identifie auprès du médiateur, ce dernier lui remet un *passe*. Le *passe* est un objet qui encapsule les données nécessaires à l'authentification de l'utilisateur après son identification auprès du système. Un *passe* sert d'identifiant pour une session de connexion et en même temps de preuve de l'identification de l'utilisateur comme *iniateur* de cette session. Les données qu'il contient sont à définir, cependant, elles doivent être suffisamment complètes pour résister à un maximum de types d'attaques. Un *passe* devra, par exemple, contenir des informations telles que l'identification de la machine depuis laquelle se connecte l'utilisateur ou la période de validité de l'identification. Le *passe* est considéré comme valide jusqu'à la déconnexion de l'utilisateur ou l'expiration d'un délai de durée de session d'utilisation. Ce délai sera spécifié par l'administrateur du médiateur.

Ensuite, toute demande effectuée par un utilisateur du système se verra munie du *passe* de celui-ci. Ces demandes concernent la consultation du catalogue des services, les appels à des services, la récupération du résultat de l'exécution d'un service et la consultation de l'état d'avancement d'une exécution d'un service. Après expiration du délai ou après la désidentification de l'utilisateur, le *passe* expiré sera, bien entendu, considéré comme invalide.

Pour prendre en compte l'ensemble des exigences concernant la sécurité intra-médiateur, il est important d'ajouter que:

- Le système de *passe* explicité plus haut sera étendu aux opérations sur les *jobs* [5] en cours.
- Ce système a l'avantage d'être aisément modifiable; seul, le *passe* doit exister. Son contenu est laissé au choix de l'implémenteur, selon ses besoins de sécurité.
- Des mécanismes de cryptage tels que SSL, PGP ou autres peuvent être également envisagés entre les clients et leur médiateur afin de sécuriser le *passe* lui-même et la phase d'identification de l'utilisateur.

La sécurité inter-médiateurs

La sécurité entre les médiateurs de la fédération est tout aussi importante que la gestion de la sécurité intra-médiateur. En effet,elle constitue une exigence non fonctionnelle capitale et c'est ce point qui conditionnera la participation de partenaires privés à la fédération.

Cette sécurité inter-médiateurs se base tout d'abord sur la mise en oeuvre d'un cryptage des communications entre chaque médiateur en utilisant, par exemple, SSL ou PGP. Ensuite, le cryptage des paramètres utilisés lors des appels aux services mais aussi le cryptage des résultats doivent intervenir entre les deux médiateurs constituant les points de départ et de fin d'une *route*. Ce cryptage est impératif pour éviter toute interception ou espionnage industriel des analyses par les tiers intermédiaires possédant des *proxies* situés sur la *route* entre le service et l'application cliente. L'utilisation d'algorithme à clés publiques et privées avec échange des clés publiques au moment de la création de *route* constitue une solution acceptable pour contrer ce problème.

Par ailleurs, la diffusion des services au travers de la fédération par un médiateur peut s'opérer à trois niveaux. Ainsi, l'autorisation d'utilisation d'un service lors de sa publication au sein la fédération sera permise au moyen de règles d'exportation. Ces règles permettront l'exportation pour un voisin direct, pour toute la fédération ou pour personne. Dans le dernier cas, l'utilisation du service est restreinte aux seuls utilisateurs du médiateur auprès duquel ils sont inscrits. Ces règles conditionneront la diffusion d'un service au travers de la fédération. En effet, un médiateur recevant un service

5. Un *job* consiste en l'exécution d'un service bioinformatique par un utilisateur.

pour son utilisation personnelle (voisin direct) ne pourra pas le proposer aux autres membres de la fédération. Bien entendu, les règles d'exportation des services seront attribuées pour chacune des connexions inter-médiateurs d'un médiateur. Ce système impose une confiance entre les propriétaires de médiateurs lorsqu'une règle limite l'exportation d'un service seulement aux voisins directs du médiateur. Aucun moyen n'est donné à son propriétaire pour s'assurer que ses voisins n'exportent pas à leur tour les informations sur le service.

Pour finir, la création d'une connexion entre deux médiateurs, quant à elle, doit s'opérer de manière bilatérale. Elle nécessite une opération sur chacun des médiateurs impliqués. Un médiateur peut toujours initier une connexion, mais elle ne sera activée que si le second médiateur l'accepte. Si une connexion est refusée ou si elle n'est pas acceptée dans un certain délai, alors elle sera supprimée. A cause de la confiance mutuelle nécessaire entre les propriétaires de deux médiateurs connectés, l'acceptation n'est pas automatique. Un intervenant humain doit donc valider chaque demande de connexion.

6.2 Maintien de la norme BSA pour l'utilisateur

Il nous semble pertinent à ce stade de notre analyse, de noter que, maintenir l'utilisation de la norme BSA pour les interactions entre utilisateurs et médiateurs devient inutile. En effet, il n'y a plus d'intérêt à considérer le médiateur lui-même comme un service BSA (Un `AnalysisService`). En effet, un ensemble de mécanismes non présents dans la norme BSA doivent être ajoutés pour répondre aux exigences. Ainsi, la recherche d'un service en catalogue, l'authentification d'un client auprès du système au moyen d'un *passe* pour chaque appel, la gestion de la récupération asynchrone des résultats et la possiblilité de créer des scripts sont autant de services différents assurés par le médiateur. Si l'on souhaite respecter strictement la norme BSA, chacun de ses services devrait être présenté à l'utilisateur sous forme d'une instance d'`AnalysisService` alors qu'ils sont normalement tous réalisés par la même entité.

Nous considérons donc que ces interactions entre utilisateurs et médiateurs se feront au moyen d'une interface définie, selon un mode strictement synchrone. Cependant, nous ferons exception quant aux types des arguments d'entrée et de résultats des appels de services qui doivent rester conformes à la norme BSA ainsi qu'aux `AnalysisTypes` retournés comme résultat de recherche dans le catalogue des services.

6.3 Les composants logiques

L'architecture que nous proposons s'articule en différentes couches offrant des fonctionnalités de plus en plus évoluées. En effet, plus la couche est d'un niveau élevé, plus sa valeur ajoutée et sa complexité sont importantes par rapport à la norme BSA de l'OMG. Ainsi, chaque couche offre un ensemble de fonctionnalités pour lesquelles, elle utilise les fonctionnalités offertes par les couches de niveau inférieur.

6.3.1 Schéma des différents composants logiques

La figure 6.5 (page 96) présente les différentes couches et composants de l'architecture du médiateur.

6.3.2 La couche 0

La couche 0 assure les fonctionnalités de base du système. Elle permet le partage des services bioinformatiques dans la fédération, la récupération de *proxies* vers ces services et la gestion des connexions entre les médiateurs. De plus, elle fournit l'accès aux ressources partagées avec les couches supérieures de l'architecture du médiateur. Normalement l'utilisateur, ou du moins l'application de l'utilisateur, n'interagit pas directement avec la couche 0.

Le ResourcesManager

Le composant *ResourcesManager*, comme son nom l'indique, prend en charge la gestion des ressources partagées au sein de la fédération. A cette fin, il s'occupe de l'enregistrement, la suppression ou les mises à jour des informations concernant les services locaux du médiateur et maintient le catalogue local des services. La prise en charge de la présence de services, de leur désinscription ou de leur mise à jour est réalisée par l'envoi de messages aux médiateurs voisins tel que décrit dans la section 6.1.1. La réception de ces messages permet la constitution du catalogue local des services. Pour la consultation de ce catalogue, le composant met à disposition une interface d'interrogation basée sur la concordance de critères de sélection fournis au moyen d'une requête et offre comme résultat les entrées correspondantes du catalogue.

Le ConnectionManager

Le *ConnectionManager* est le composant qui gère les connexions du médiateur avec les autres médiateurs auxquels il est connecté. Il maintient une liste des médiateurs auxquels il est connecté; il se charge également de vérifier leurs états et d'effecteur les demandes de fermeture ou d'ouverture de

connexions. Il fournit aux autres composants de la couche 0 les références vers les composants des médiateurs voisins lorsque cela s'avère nécessaire. Dans le cas d'une communication directe entre les médiateurs, ces références donnent directement accès aux composants des médiateurs distants. Dans le cas de communications inter-médiateurs indirectes, ces références donnent accès à des composants façades qui gèrent les aspects complexes de ce type de communication (voir section 6.4).

Le ProxiesManager

Le *ProxiesManager* gère le cycle de vie des *proxies* créés au sein du médiateur lors des appels aux services bioinformatiques. Lorsqu'un nouveau *proxy* est nécessaire, le *ProxiesManager* interagit avec les autres composants afin de le créer. Le *ProxiesManager* est donc un composant de type `factory` [GHJV94] de *proxies*. Il surveille l'état des *proxies* créés et peut les supprimer ou les mettre en veille si nécessaire. Il sert d'interface entre la couche 0 et la couche 1 pour tout ce qui touche à la manipulation de *proxies*.

Le LoadBalancing

Lorsque plusieurs *routes* sont disponibles pour accéder à un service, le *LoadBalancing* a la responsabilité du choix de la *route* à utiliser. L'utilisation d'un composant distinct pour cette tâche permet d'assurer une plus grande flexibilité dans le choix des *routes*. En effet, en le séparant des autres composants, on assure son évolutivité indépendamment des autres composants de la couche 0. Ainsi, l'implémenteur est libre de ses choix quant à sa gestion de la répartition de la charge et de sa perception de la *meilleure route* (la moins chère, la plus rapide, etc.).

L'AuditManager couche 0

L'*AuditManager* a pour tâche de gérer une trace de tous les événements et activités survenus en couche 0. Ses traces peuvent servir pour la maintenance du système mais aussi pour affiner les processus de sélection des *routes* du *LoadBalancing* ou pour fournir des informations pour une facturation ultérieure des utilisations de services aux utilisateurs de la couche 1. Ce composant fournit une interface de création de traces. Celle-ci est utilisée par les autres composants de la couche 0. La responsabilité de la création des traces revient donc aux composants de la couche. Mais des mécanismes reposant sur les détails d'implémentation peuvent également être utilisés par l'*AuditManager*, lui-même, pour générer des traces. L'*AuditManager* fournit également une interface de consultation des traces aux autres composants. Bien entendu, les interfaces de création et de consultation doivent reposer sur une même standardisation de leur format.

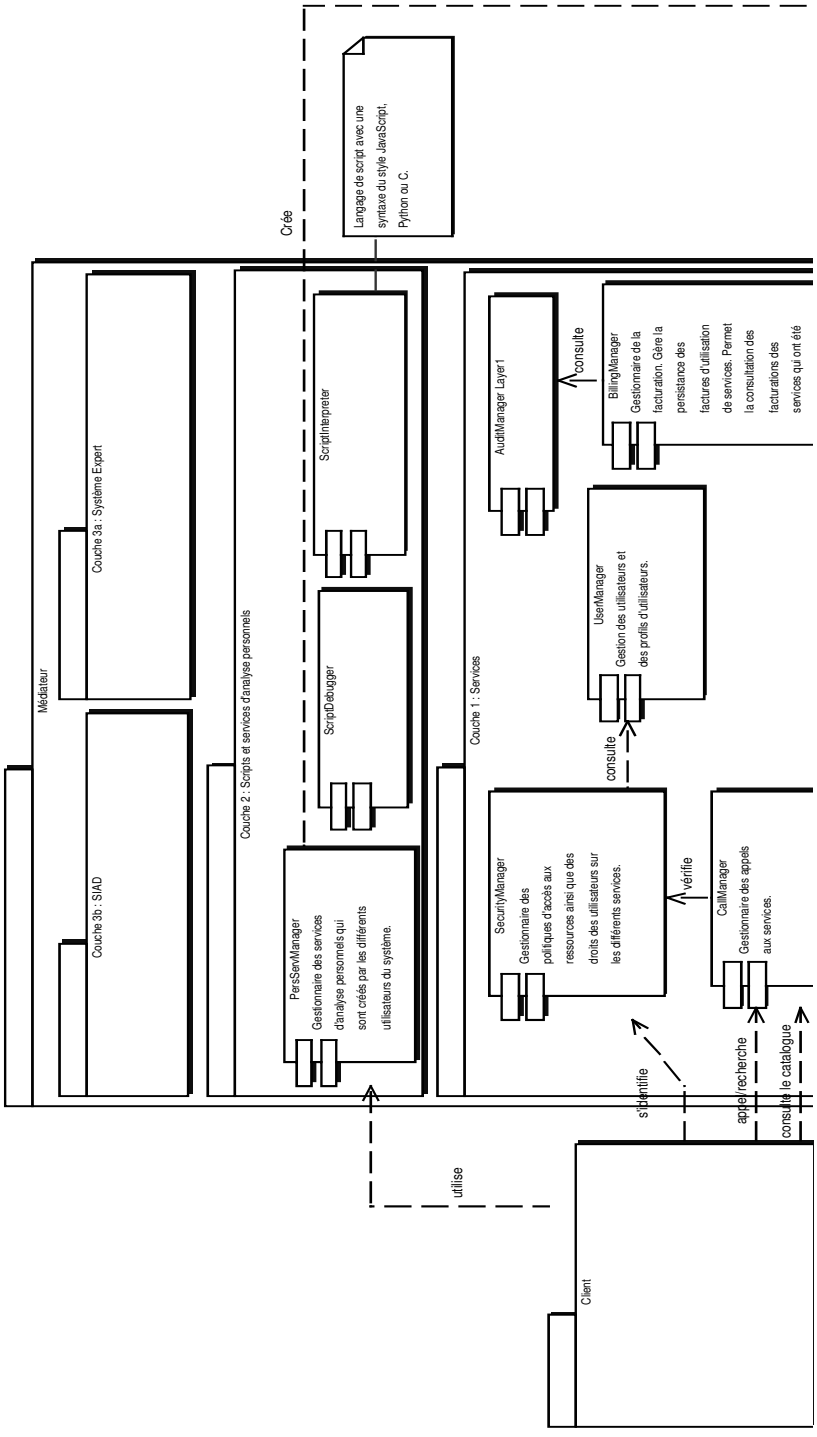


FIG. 6.5 – *Diagramme de composants: Architecture logique en couches*

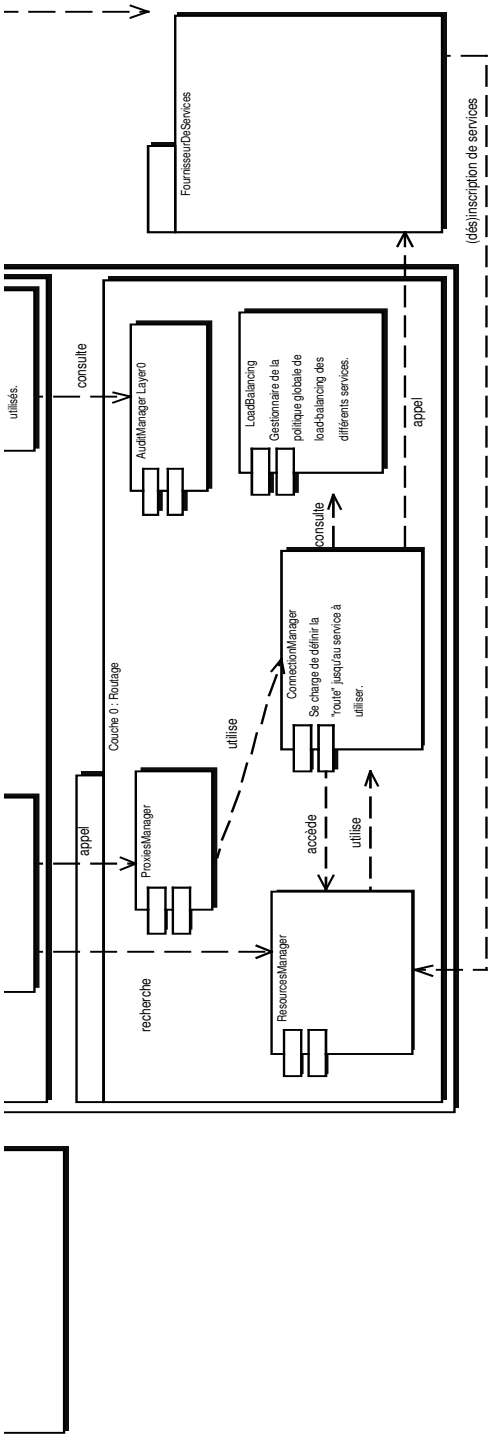


Figure 6.5 (suite)

6.3.3 La couche 1

La couche 1 assure aux utilisateurs du système la fonction de *proxy* vis-à-vis de la couche 0. En effet, elle permet aux utilisateurs d'employer des services BSA sans se préoccuper des mécanismes de distribution confiés à la couche 0 et du respect du diagramme de séquence imposé par la norme. Elle assure aussi les mécanismes de sécurité intra-médiateur et la gestion asynchrone des appels de services pour le client (récupération ultérieure des résultats, etc.).

Le UserManager

Le *UserManager* prend en compte la gestion des utilisateurs du système. Il assure avec le composant *SecurityManager* la sécurité intra-médiateur. Cette tâche de gestion des utilisateurs a été séparée de la gestion de la sécurité pour permettre l'intégration, au sein d'un médiateur, de systèmes existants assurant le même rôle(serveur `NIS`, serveur `LDAP`, bases de données propriétaires d'utilisateurs, etc.). Ce composant joue un rôle de *pont* entre un système de gestion existant et les composants du médiateur en le présentant comme une interface clairement définie.

UserManager prendra en compte les exigences fonctionnelles et non fonctionnelles telles que la vérification de l'appartenance d'un utilisateur au système, l'ajout d'utilisateurs ou de groupes, la suppression d'utilisateurs ou de groupes, la modification des diverses appartenances aux groupes ou profil, l'association d'un utilisateur à un groupe ou à un profil, ou encore l'obtention des informations concernant un utilisateur.

Le SecurityManager

Le *SecurityManager* assure l'identification de l'utilisateur dans le système et la gestion de ses droits sur les services. A cette fin, il permet l'association de droits entre un service bioinformatique et un utilisateur, un groupe ou un profil. Par ailleurs, il régule aussi l'accès au système en imposant l'identification de l'utilisateur. Celle-ci s'effectue au moyen d'un *login* et d'un *mot de passe*. Suite à une identification correcte, le système remet à l'utilisateur un *passe*. Ce *passe* constitue un *laisser-passer* pour l'utilisateur. Ce *laisser-passer* identifie l'utilisateur pour toutes ses actions pour un certain laps de temps. Ce composant de gestion de la sécurité associe également un droit d'accès aux éventuels *jobs*[6] en cours d'exécution.

Le CallManager

Le *CallManager* joue le rôle de *proxy* entre le client et la couche 0. L'utilisateur s'adresse à celui-ci pour réaliser un appel à un service, obtenir son

6. Un *job* consiste en l'exécution d'un service bioinformatique par un utilisateur.

résultat, vérifier l'état d'avancement d'un *job* et consulter le catalogue des services. Ce *proxy* vis-à-vis des *proxies* de services permet un fonctionnement asynchrone de l'application cliente. C'est ce composant qui gère les appels BSA sur toute leur durée et qui délivre des *tickets* (identifiants) de *job* aux utilisateurs, leur permettant de vérifier l'état d'exécution de services ou d'obtenir leurs résultats *a posteriori*.

L'AuditManager de la couche 1

L'*AuditManager* en couche 1 joue le même rôle que le composant *AuditManager* en couche 0 mais vis-à-vis des composants de la couche 1: à savoir, les tâches de création, de maintien et d'accès aux traces d'événements au sein de la couche.

Le BillingManager

Le composant *BillingManager* répond à l'exigence demandant la possibilité de facturer l'utilisation d'un service aux utilisateurs (voir section 5.2 page 65) du système qu'ils soient locaux ou non. Pour y parvenir, le composant *BillingManger* se base sur les traces enregistrées par les composants d'audit indiquant les utilisations de services les facturer aux utilisateurs (utilisateurs locaux ou médiateurs partenaires) du système.

6.3.4 La couche 2

La couche 2 assure les fonctionnalités d'automatisation et d'utilisation *avancée* et *experte* du système (scripts, sessions d'utilisation, etc.) remarquées dans les exigences fonctionnelles. Elle fournit ainsi les outils aux utilisateurs de profil avancé et expert.

La gestion des scripts d'analyse personnels

Ce composant a pour rôle d'assurer la gestion des scripts d'analyse personnels. Ceux-ci sont automatiquement générés à la fin de chaque session d'utilisation de service et permettent de *rejouer* cette session. De plus, à chaque exécution de service entièrement effectuée, un protocole complet détaillant tous les paramètres d'exécution (valeurs des arguments, valeurs du résultat, version des programmes ou des bases de données utilisées, localisation du service, etc.) est émis. Cette exigence fonctionnelle est très importante pour le biologiste car il est soucieux de connaître tous ces paramètres. Dans la pratique, il n'est pas rare qu'un changement de version du programme (basé fréquemment sur des heuristiques) ou de la base de données utilisée modifie les résultats obtenus.

Les opérations suivantes de gestion des scripts seront prises en compte:

- **Enregistrer:** permet l'enregistrement d'un script d'analyse personnel généré lors d'une exécution d'un service, ou d'un script écrit par un l'utilisateur.
- **Supprimer:** permet de supprimer un script d'analyse personnel.
- **Renommer:** permet de modifier la dénomination d'un script d'analyse personnel qui a été précédemment enregistré.
- **Modifer/Editer:** permet de modifier le code d'un script d'analyse personnel.
- **Créer:** permet de créer un nouveau script d'analyse personnel.
- **Changer les droits d'accès/partager:** permet de modifier les droits d'accès à un script d'analyse personnel. Ainsi, l'utilisateur peut donner des droits d'accès à d'autres utilisateurs du médiateur mais aussi à des groupes ou des profils de son choix. Cette modification des droits doit cependant être avalisée par l'administrateur du médiateur. Ce dernier accepte la modification des droits en fonction de la charge actuelle du système et de la charge additionnelle que risque d'entraîner cette modification.

L'interpréteur et le débugger de scripts

Ces composants ont pour but d'offrir un interpréteur pour les scripts d'analyse personnels pour les exécuter ainsi qu'un outil de "débuggage" pour aider le concepteur de scripts dans sa tâche de conception.

6.3.5 La couche 3

La couche 3 assure les fonctionnalités dites d'intelligence du système. Cette couche apporte la plus grande valeur ajoutée en terme d'automatisation des services.

Sous-couche SIAD

Un système d'information d'aide à la décision peut être défini comme suit: *"Un système d'information mettant en oeuvre des technologies d'information et utilisé comme support de la prise de décisions relatives à des situations où il n'est ni possible, ni désirable d'automatiser l'entièreté du processus de décision."* [Jac02]

Ainsi, les trois règles d'or pour un SIAD telles que proposées par [Jac02] sont: *"Un SIAD est une aide pas un substitut, un SIAD doit être personnalisé au contexte du processus de décision et un SIAD doit être compatible avec le style de décision des utilisateurs."*. Le rôle de cette sous-couche est d'offrir des outils d'aide à la décision pour les bioinformaticiens.

Sous-couche système expert

Un système expert peut être défini comme suit: *"Un système expert consiste en un environnement logiciel/matériel capable de résoudre des problèmes requérant une grande quantité de connaissances (de savoir-faire, d'expertise), d'acquérir de nouvelles connaissances en vue d'étendre ou d'adapter leur expertise et d'expliquer leur raisonnement."* [Jac02].

Ainsi le rôle de cette sous-couche est d'offrir des outils intelligents autonomes à haute valeur ajoutée.

6.4 Modèles de communication inter-médiateurs

6.4.1 Composants communiquant directement

Du système proposé par [DD03], point de départ de notre réflexion, nous avons conservé l'indirection de requêtes vers un service bioinformatique. Cette indirection est réalisée par un *chaînage de proxies* BSA. Chaque médiateur, se trouvant sur la *route* entre le médiateur client du service et le médiateur fournisseur du service, contient un *proxy* dont la tâche principale est d'assurer la redirection des appels et des réponses du client vers le service et vice versa. Cette situation est représentée sur la figure 6.6, ou l'on peut voir les différents messages entre client et médiateur (flèches et lignes en pointillé).

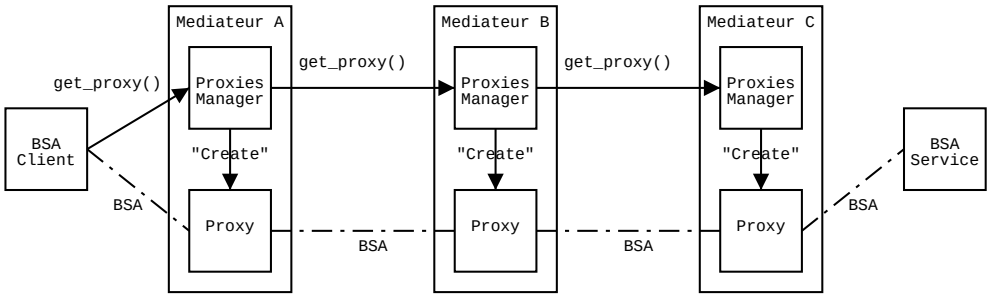


FIG. 6.6 – *Composants communiquant directement*

Ce système impose des communications entre les composants (*proxies* et autres) des différents médiateurs de la fédération et prévoit que elles soient directes entre les composants. Ainsi, lorsqu'un médiateur a établi une connexion avec un autre, au moyen de son *ConnectionManager*, les autres composants des médiateurs s'échangent directement des messages au moyen d'un middleware, CORBA par exemple (voir figure 6.7).

Cependant, il est plus que probable que les médiateurs soient déployés sur des sites physiquement fort éloignés et administrés par des organisations

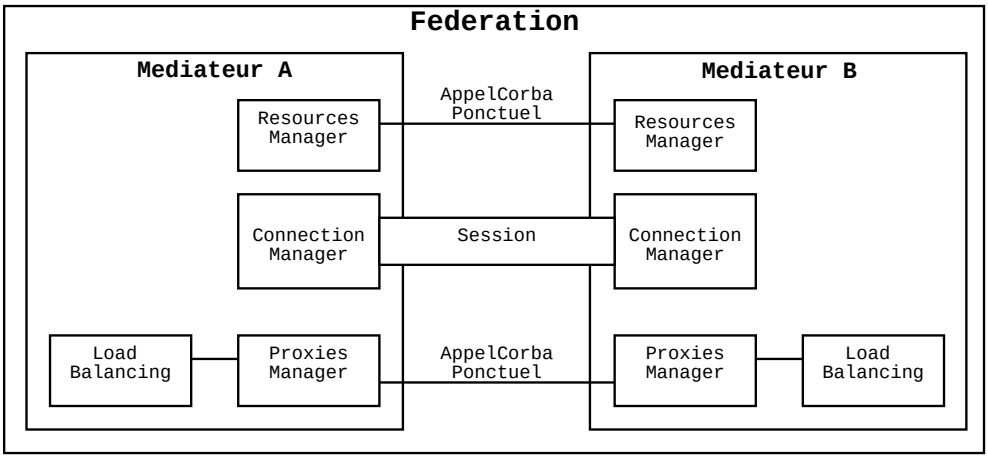


FIG. 6.7 – *Communications directes entre composants*

différentes. Cette forte distribution physique impliquera de nombreuses communications entre objets distribués dont les hôtes seront éventuellement protégés ou masqués par des *Firewalls* ou des *Routers NAT*[7]. Or, dans ce cas de figure, les communications de certains *middlewares* (CORBA notamment) ne sont pas aisées voire impossibles.

6.4.2 Composants communiquant au travers d'un composant de communication

De manière à régler le problème de communication évoqué dans le point précédent, nous avons imaginé de charger un seul composant de la gestion de toutes les communications entre les médiateurs. Ce composant servirait d'interface entre les composants de son médiateur et ceux des médiateurs auxquels il est connecté. La figure 6.8 illustre les communications entre les composants de deux médiateurs utilisant ce principe. Les *ConnectionManagers* maintiennent les sessions entre les médiateurs et offrent aux composants de leur propre médiateur une interface, sorte de voie d'accès, vers les composants de l'autre médiateur.

L'utilisation d'un composant de communication pour les communications entre composants de différents médiateurs remet en question le principe de *chaînage de proxies*. En effet, au sein d'un médiateur intermédiaire, on assisterait à une sorte d'effet de "ping-pong" entre les *proxies* et le composant de communication. La figure 6.9 montre, par un diagramme de séquence, ces échanges. On peut y voir qu'un composant de communication recevant un message d'un médiateur précédent sur la *route*, le redirige vers le *proxy* as-

7. Network Address Translation, système permettant à plusieurs ordinateurs différents d'accéder à l'Internet en utilisant une seule adresse IP publique.

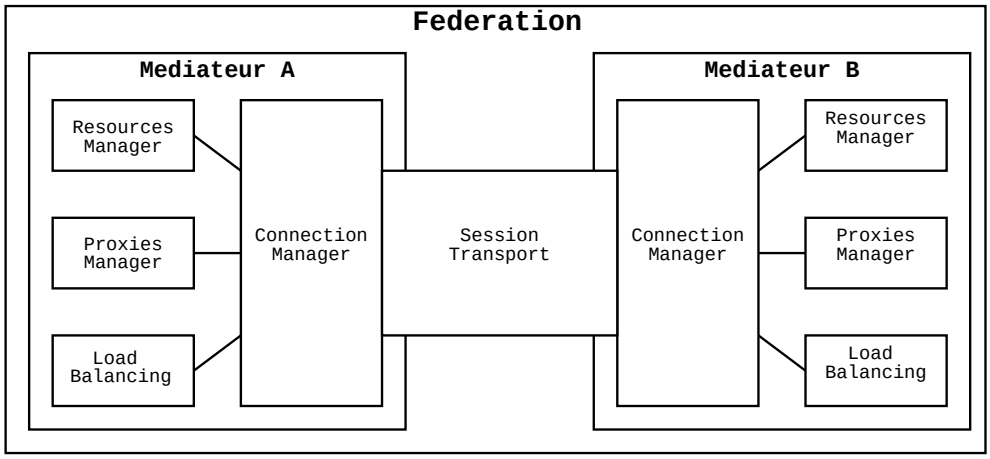


FIG. 6.8 – *Communications via un composant de communication*

socié qui le retransmet au composant de communication pour le faire suivre vers le *proxy* suivant dans la *chaîne*, situé au sein d'un autre médiateur. Il serait plus logique que ce soit les composants de communication qui redirigent eux-mêmes les messages directement vers le médiateur suivant, sans passer à chaque fois par un *proxy*, comme indiqué sur la figure 6.10.

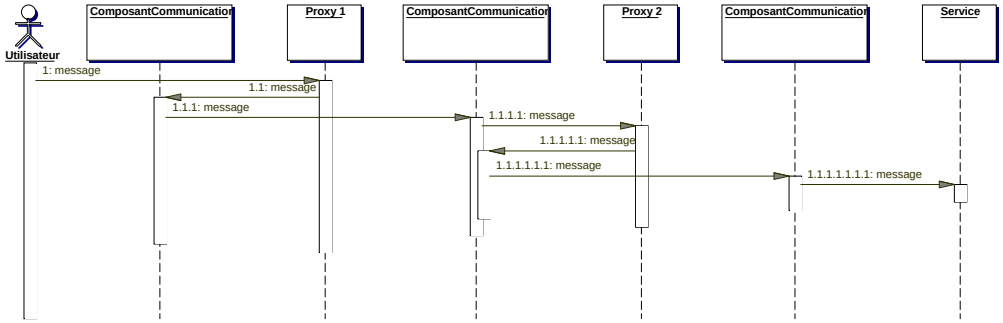


FIG. 6.9 – Chaîne de proxies *et effet de "ping-pong" entre composants.*

Avec un tel système, on limite l'utilisation de la norme BSA aux seules relations *client(CallManager)-proxy* et *proxy-service*. Ceci devrait permettre l'ouverture du système au partage d'autres ressources que des ressources BSA. Le routage des messages, quant à lui, se ferait au moyen d'un *forwarding* entre *ConnectionManager*. Par ailleurs, ce système assurerait plus facilement la sécurité au niveau des liens entre médiateurs[8] et éviterait les

8. Le fonctionnement au moyen d'un composant de communication offre plus de sou-

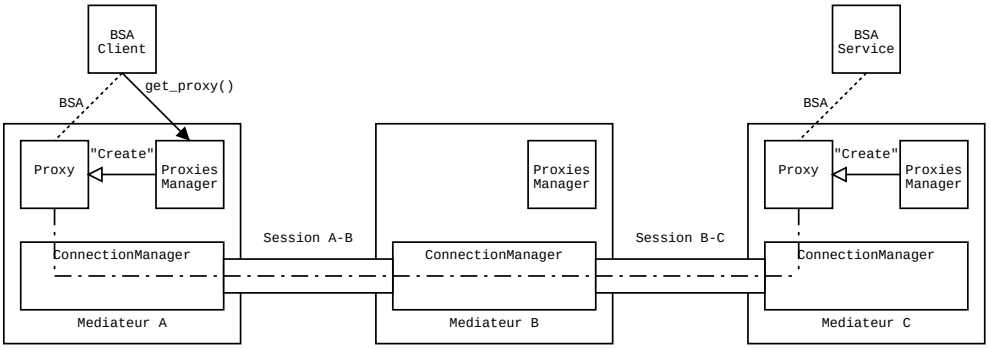


FIG. 6.10 – *Transmission des messages BSA sans* chaîne de proxies.

problèmes liés au déploiement (Firewalls, etc.).

6.4.3 Notre choix

La solution de communication entre composants inter-médiateurs via un composant de communication constituerait le choix idéal. En effet, elle semble la plus porteuse en terme de sécurité et de charge du système car elle rendrait possible la création d'une extension du composant *Connection-Manager* permettant un cryptage des communications inter-médiateurs ou l'utilisation d'autres modes de communication que ceux prévus initialement (SOAP, protocole dédié, etc.). De plus, cette solution possède l'avantage d'alléger le système puisque tous les médiateurs d'une *route* entre le fournisseur et l' utilisateur d'un service ne se voient plus obligés de maintenir un *proxy* qui soit un composant BSA complet.

Nous avons cependant décidé d'utiliser, dans un premier temps du moins, la première solution: les communications directes entre composants. Bien que cette solution ne présente pas tous les avantages de la solution offerte par un composant de communication, elle est plus simple à mettre en oeuvre. En effet, la création et la gestion des *chaînes de proxies* sont beaucoup plus simples d'un point de vue programmation que la gestion d'un composant de communication.

Cependant, nous n'estimons pas que cette décision constitue une barrière à l'évolution future de l'architecture vers un système de communication au moyen d'un composant de communication. Un tel système pourrait être mis en place assez facilement en apportant quelques modifications au fonctionnement des *proxies* situés au sein de chaque médiateur ainsi qu'au composant *ConnectionManager*. Dans le mode de communication se faisant au moyen d'un composant de communication, les *proxies* joueraient leur rôle d'inter-

plesse pour gérer un éventuel cryptage des messages échangés entre deux points extrêmes d'une *route*.

face entre le client et le service et n'assureraient plus le transport. Celui-ci serait délégué entièrement au composant *ConnectionManager*. Le composant *ConnectionManager*, quant à lui, maintiendrait des composants façades des composants des autres médiateurs auxquels il est connecté. Cette "transformation" serait opérable assez facilement dans l'architecture actuelle ainsi que dans notre implémentation car le *ConnectionManager* y est le gardien des références des composants distants. Il est donc libre de fournir aux composants de son médiateur, des références vers des composants façades qu'il gérerait lui-même plutôt que des références réelles vers des composants distants.

La figure 6.11 illustre une telle configuration, le *ConnectionManager* du médiateur A maintient des composants façades, *Proxy RM B* et *Proxy PM B*, vers les composants du médiateur B, *ResourcesManager B* et *ProxiesManager B*. Le *ConnectionManager* du médiateur B fait de même pour les composants de A. Lorsqu'un composant du médiateur A désire envoyer un message à un composant du médiateur B, il le communique au composant façade correspondant situé dans le *ConnectionManager* de A (1). Le *ConnectionManager* du médiateur A prend alors en charge le message et le transmet au *ConnectionManager* du médiateur B (2). Enfin le *ConnectionManager* du médiateur B le transmet au composant de destination (3).

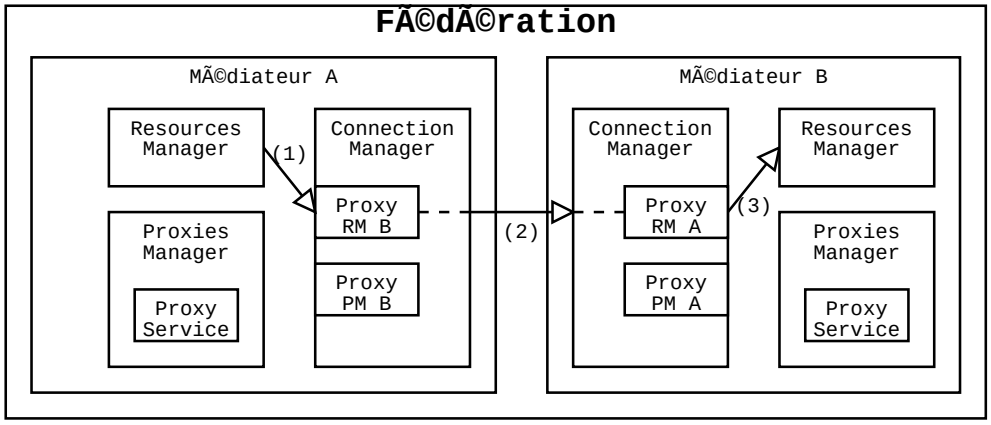


FIG. 6.11 – *Evolution d'un système de communication directe vers un système via un tunnel*

6.5 Dynamique de la couche 0 du système

Cette section a pour but d'illustrer visuellement la dynamique des composants de la couche 0 du système au moyen de diagrammes de séquence.

6.5.1 Topologie des connexions

Pour illustrer les interactions entre composants, nous utilisons une topologie d'exemple telle que présentée dans la figure 6.12. On y retrouve deux services et quatre médiateurs. Le premier service est connecté au premier médiateur, tandis que le second service est connecté au troisième médiateur. Le premier et le troisième médiateur sont tous deux connectés au second et au quatrième médiateur.

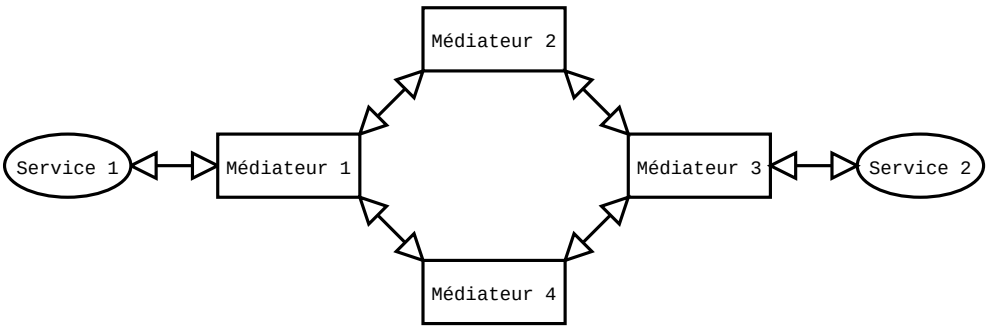


FIG. 6.12 – *Topologie des connexions*

On peut retrouver dans cette topologie les cas suivants:

1. Un médiateur et un service local, soit le médiateur 1 et le service 1.
2. Deux médiateurs et un service, soit les médiateurs 1 et 2 et le service 1. Le service 1 est alors un service distant accessible par le médiateur 2.
3. Trois médiateurs et un service, soit les médiateurs 1, 2 et 3 et le service 1. Le service 1 est alors un service distant accessible par les médiateurs 2 et 3.
4. Trois médiateurs et deux services, soit les médiateurs 1, 2 et 3 et les services 1 et 2. Les services sont alors des services distants accessibles par le médiateur 2.
5. Quatre médiateurs et deux services, soit tous les éléments présents sur la figure 6.12. Les médiateurs 2 et 4 ayant alors, pour chaque service, deux chemins d'accès possibles. On retrouve aussi la possibilité d'une boucle dans cette topologie.

Ces différents cas permettent d'illustrer un maximum de cas particuliers et de situations "courantes" de l'utilisation du système.

6.5.2 (Dés)Inscription d'un service de la fédération

La figure 6.13 illustre les interactions entre composants lors de l'inscription, la désinscription et la mise à jour d'un service auprès du *Resources-*

Manager. Pour illustrer ces interactions, nous plaçons dans le cas “Deux médiateurs et un service” (2).

Inscription d’un nouveau service

L’administrateur enregistre un nouveau service (le service 1) auprès du *ResourcesManager* du médiateur 1. Le *ResourcesManager* du médiateur 1 transmet ensuite les informations concernant le service au *ResourcesManager* du médiateur 2. Cette transmission se réalise au moyen des composants *ConnectionManagers* qui assurent soit la communication entre les médiateurs, soit la gestion des références des objets distants des composants de l’autre paire de la connexion.

Mise à jour de la description d’un service

L’administrateur met à jour les informations d’un service auprès du *ResourcesManager* du médiateur 1. Le *ResourcesManager* du médiateur 1 transmet ensuite les informations mises à jour au *ResourcesManager* du médiateur 2. Cette transmission se réalise au moyen des composants *ConnectionManagers* qui assurent soit la communication entre les médiateurs, soit la gestion des références des objets distants des composants de l’autre paire de la connexion.

Désinscription d’un service

L’administrateur désinscrit un service auprès du *ResourcesManager* du médiateur 1. Le *ResourcesManager* du médiateur 1 transmet ensuite la désinscription du service au *ResourcesManager* du médiateur 2. Cette transmission se réalise au moyen des composants *ConnectionManagers* qui assurent soit la communication entre les médiateurs soit la gestion des références des objets distants des composants de l’autre paire de la connexion.

S’il y a plus de médiateurs dans la fédération, les mêmes interactions ont lieu de proche en proche. Un médiateur qui reçoit un ajout, une mise à jour ou une désinscription de service le transmet à ses voisins . Bien entendu, il ne modifie pas les informations reçues mais il vérifie leur validité et s’assure qu’il a le droit de les transmettre (en accord avec les principes de fonctionnement exposés dans les sections 6.1.1 et 6.1.3).

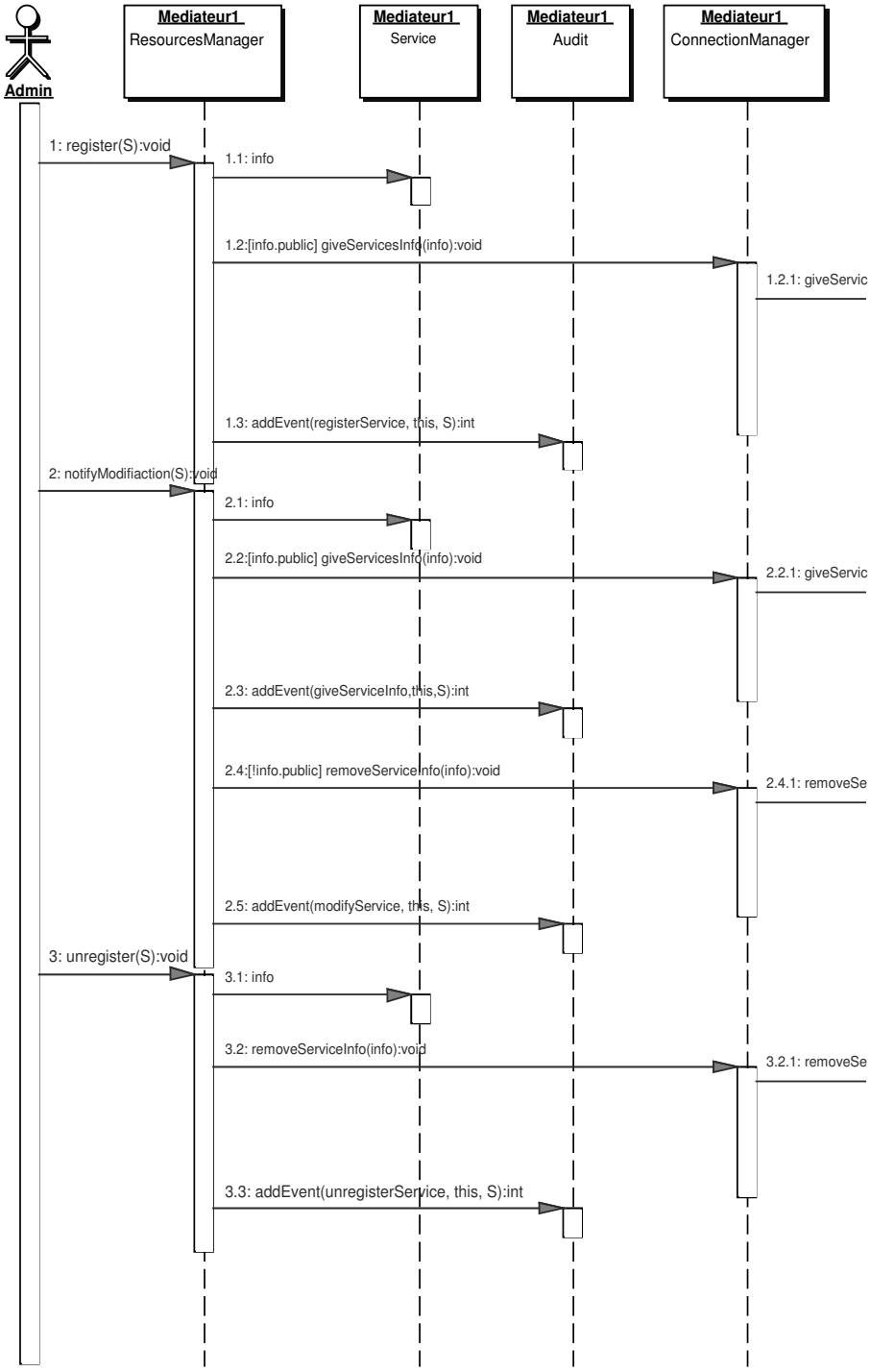


FIG. 6.13 – *(Dés)Inscription et mise à jour d'un service de la fédération*

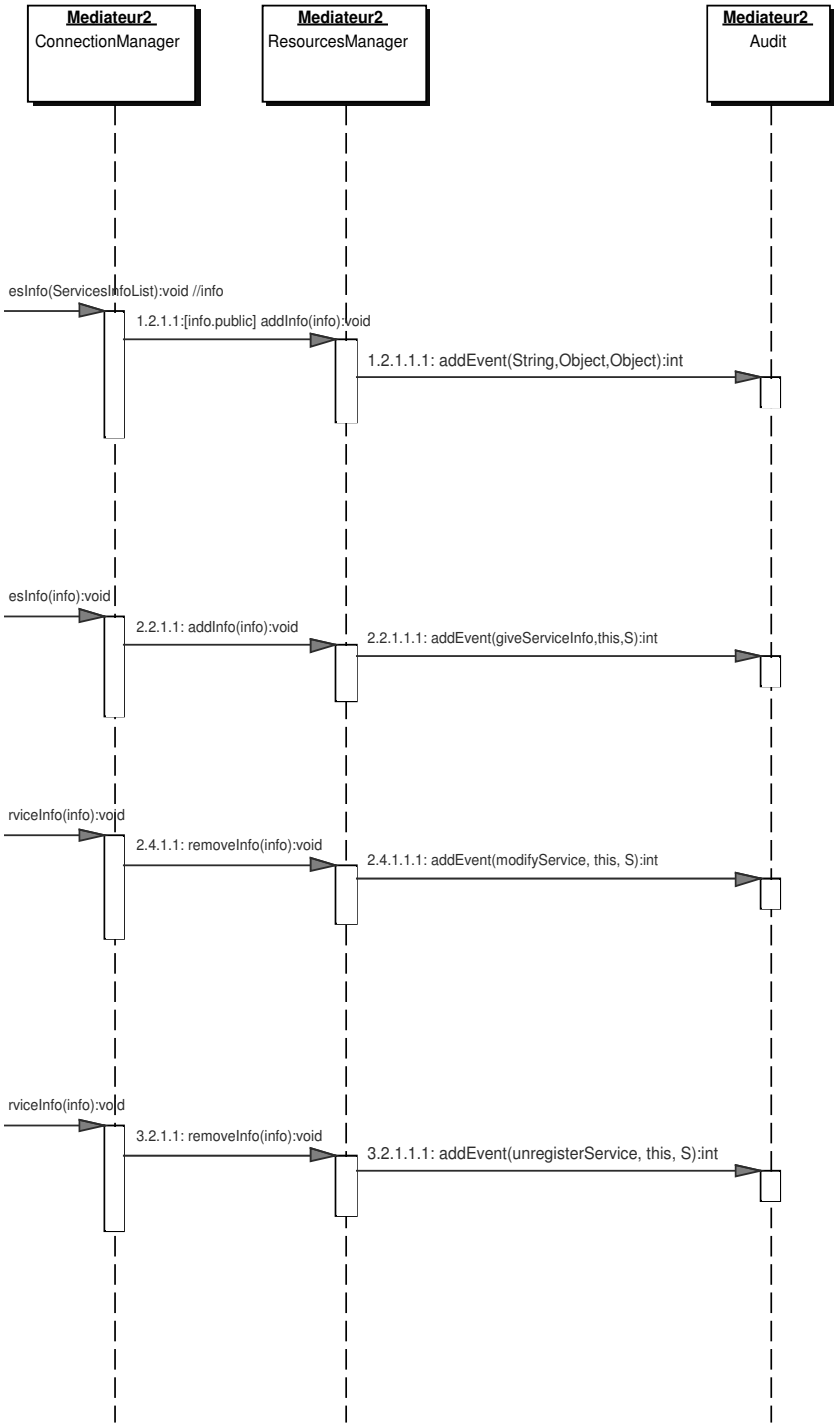


Figure 6.13 (suite)

6.5.3 Appel d'un service local à un médiateur

La figure 6.14 illustre les interactions entre composants lors d'un appel à un service local d'un médiateur, c'est-à-dire l'appel d'un service connecté au même médiateur que le client. Pour illustrer ces interactions, nous plaçons dans le cas "Un médiateur et un service" (1).

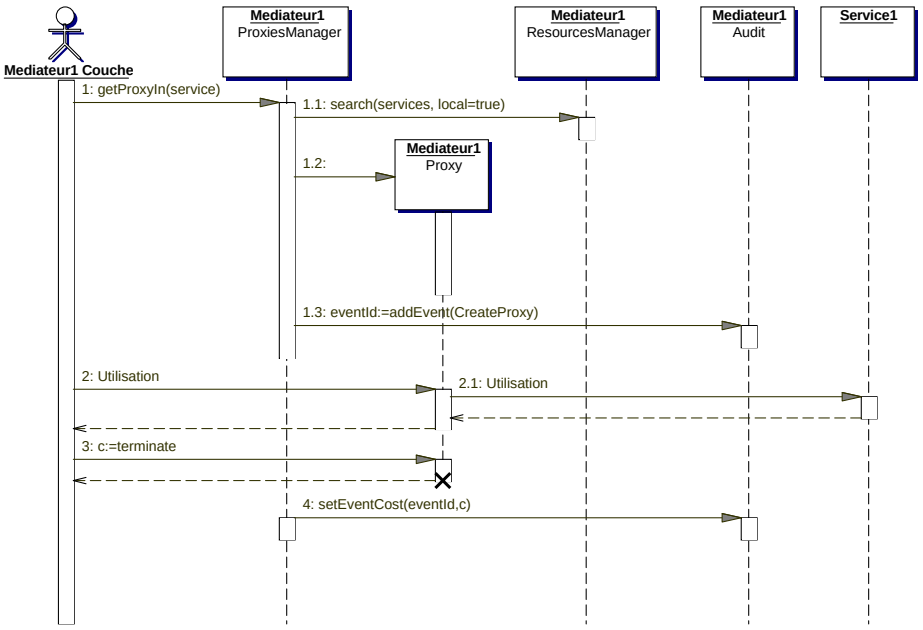


FIG. 6.14 – *Appel d'un service local à un médiateur*

La couche 1 demande au *ProxiesManager* de lui fournir un *proxy* vers le service désiré. Le *ProxiesManager* vérifie que le service est un service local auprès du *ResourcesManager* (paramètre `local=true` de l'opération `search()`). Dans ce cas, il fournit à la couche 1 la référence d'objet distant d'un *proxy* vers le service qu'il vient de créer. Il enregistre également auprès du composant *AuditManager* une information de début d'utilisation. Si le service n'est pas local, un appel distant sera effectué (voir section 6.5.4).

Lorsque la couche 1 utilise le *proxy* (implémentant les interfaces `AnalysisService`, `AnalysisInstance` et `JobControl` de la norme BSA), les appels de méthodes sont reportés jusqu'au service. L'opération `Utilisation` sur le diagramme de séquence représente les interactions entre les clients et le service telles que définies dans la norme BSA.

Lorsque la couche 1 a terminé son appel au service, le *ProxiesManager* détruit le *proxy* du service et enregistre une information de fin d'utilisation auprès du composant *AuditManager*.

6.5.4 Appel d'un service distant d'un médiateur

Les figures 6.15 et 6.16 illustrent les interactions entre composants lors d'un appel à un service distant d'un médiateur. Pour illustrer ces interactions, nous nous plaçons dans les cas "Deux médiateurs et un service" (2) et "Trois médiateurs et un service" (3).

Pour le diagramme de séquence de la figure 6.15, nous nous situons dans le cas où l'appel est réalisé sur un service inscrit auprès du voisin direct du médiateur auquel est connecté le client. La figure 6.16 illustre les interactions entre composants lors d'un appel à un service distant d'un médiateur lorsque trois médiateurs sont impliqués.

Le fonctionnement est sensiblement le même qu'un appel local. La différence, se situe après la vérification effectuée par le *ResourcesManager* local. En effet, le service n'étant pas local, le *ProxiesManager* du médiateur 2 opère une recherche de *routes* auprès des médiateurs auxquels il est connecté (dans les deux cas, seulement le médiateur 1). Il reçoit un ensemble de *routes* candidates (dans les deux cas, il ne reçoit qu'une seule *route*). Un fois qu'il en aura choisi une, il va l'*ouvrir* via le premier médiateur de la *route* choisie et ainsi déclencher la création de la *chaîne de proxies*. Le principe est récursif pour chaque médiateur présent sur la *route*, comme l'illustre le second diagramme (figure 6.16). Chaque *route* sera identifiée par une chaîne d'identifiants insérés par chaque médiateur lors de la recherche de la *route*.

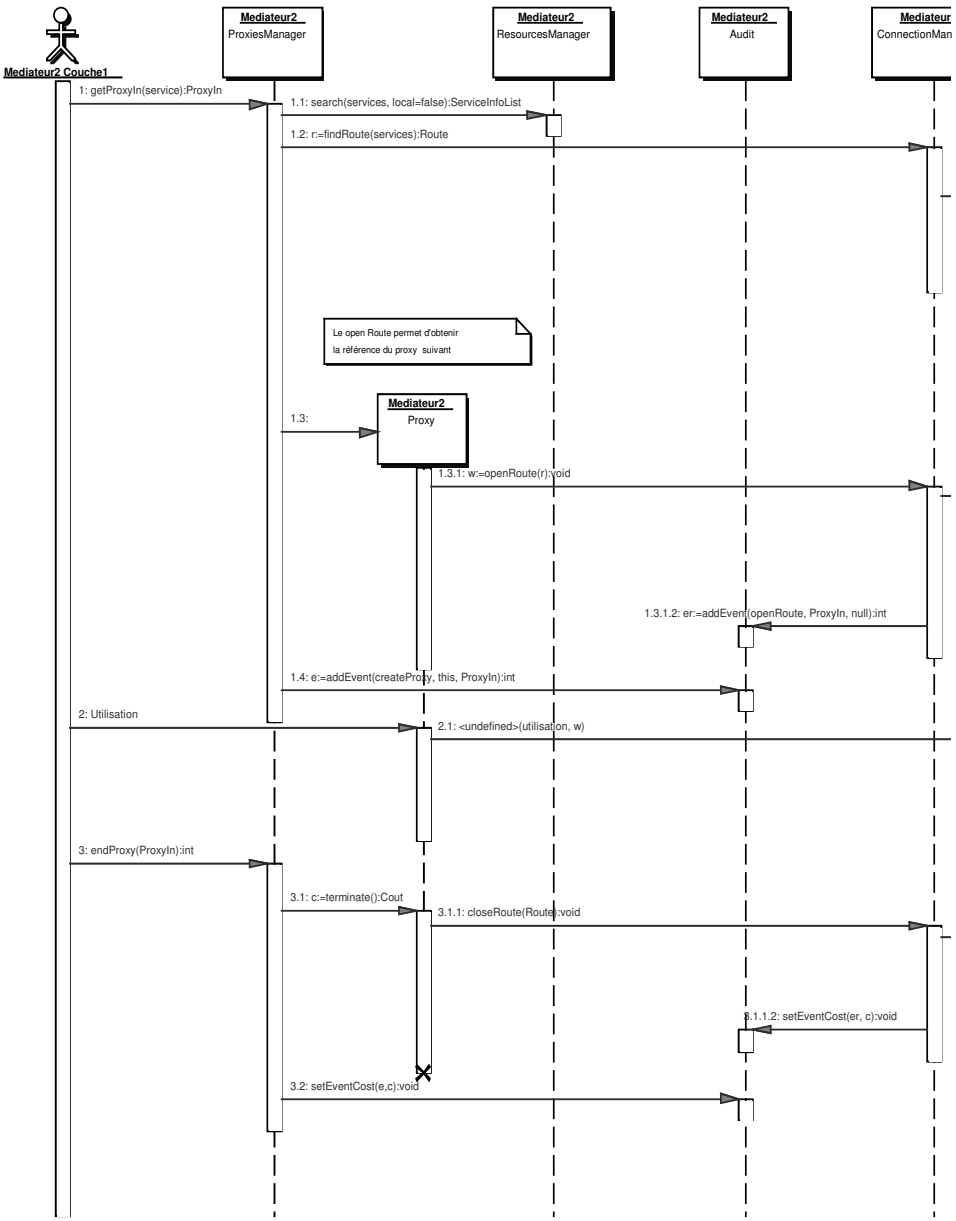


FIG. 6.15 – *Appel d'un service distant dans le cas "Deux médiateurs et un service"*

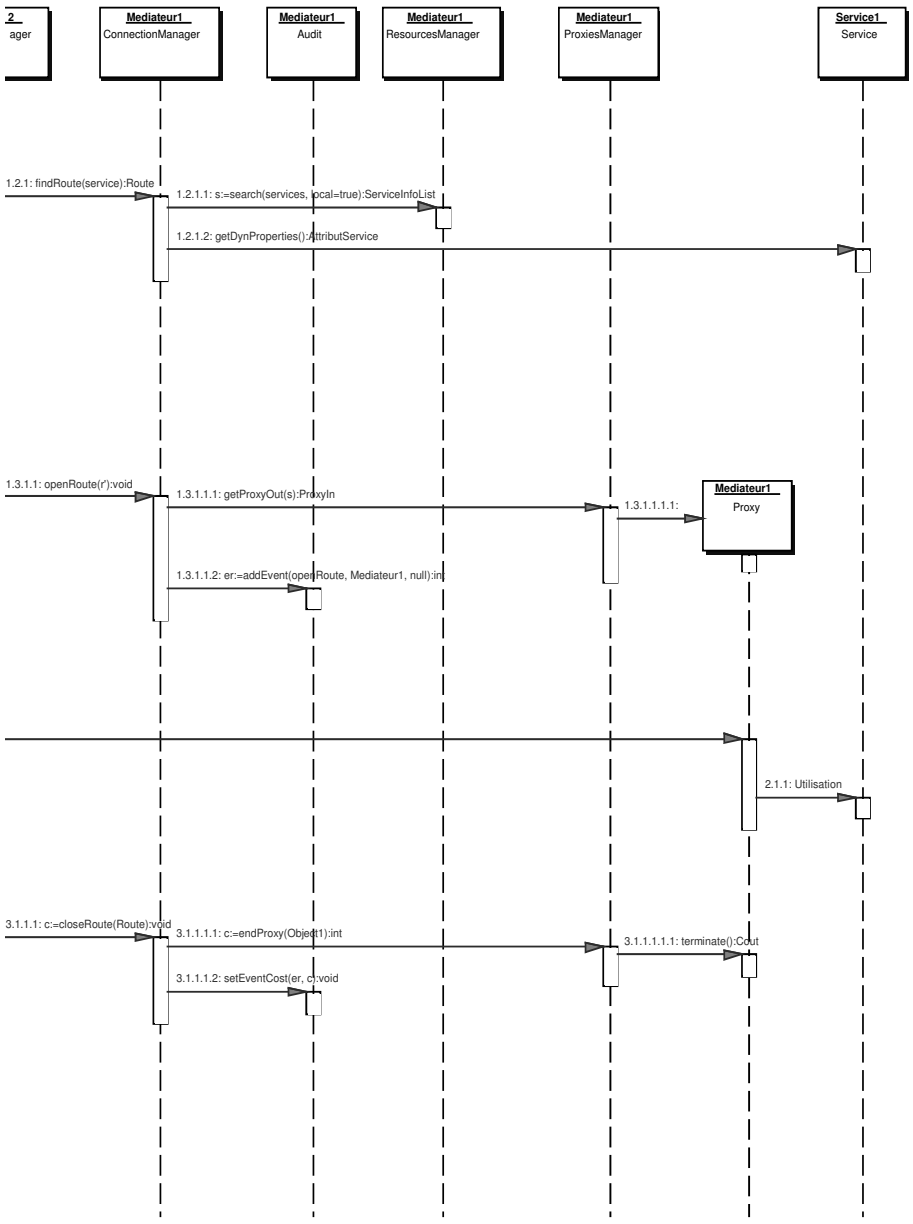


Figure 6.15 (suite)

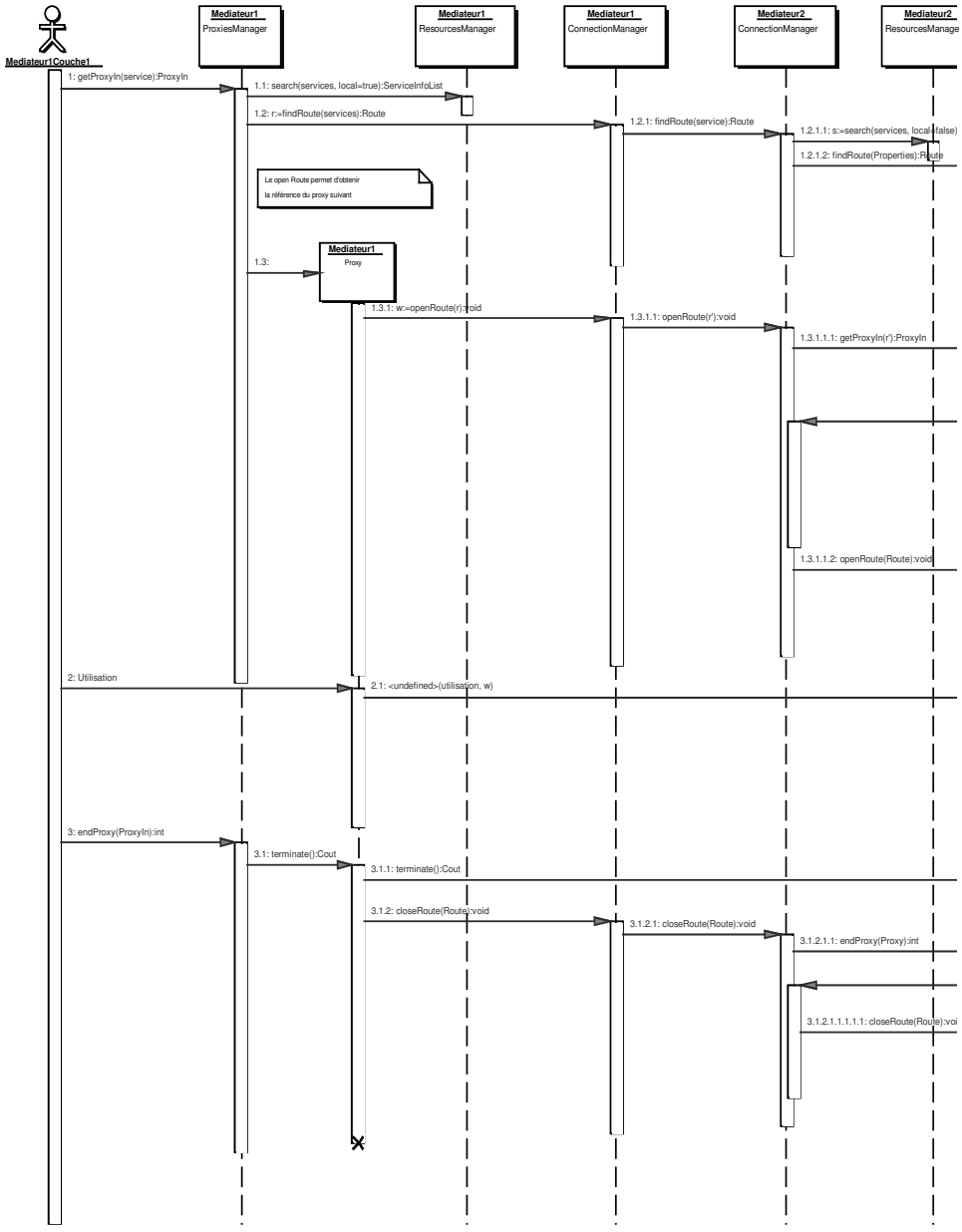


FIG. 6.16 – *Appel d'un service distant dans le cas "Trois médiateurs et un service"*

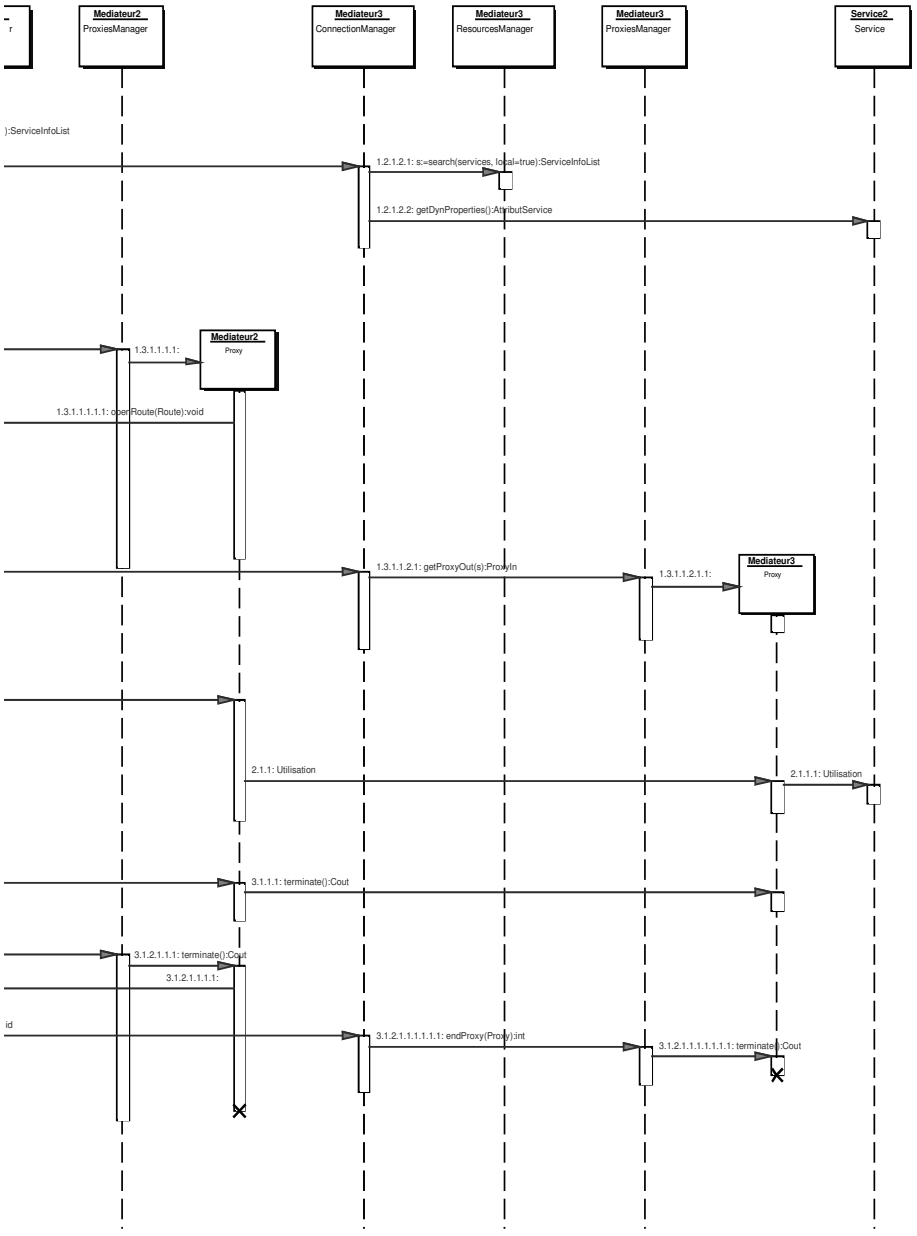


Figure 6.16 (suite)

6.5.5 Sélection et recherche d'une *route* lors d'un appel

La figure 6.17 illustre les interactions entre composants lors de la recherche et de la sélection d'une *route*. Nous nous situons dans le cas "Trois médiateurs et deux services" (4).

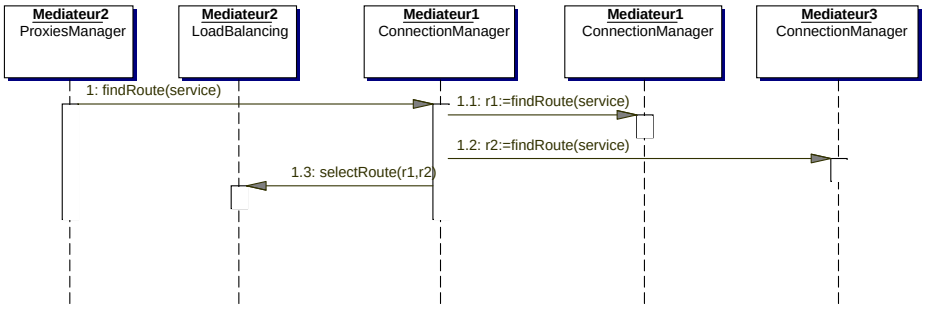


FIG. 6.17 – *Sélection et recherche d'une* route *lors d'un appel dans le cas "Trois médiateurs et deux services"*

Le choix d'une *route* parmi l'ensemble des candidates reçues par le *ConnectionManager* est effectué par le composant de *LoadBalancing*.

6.5.6 Ajout d'une connexion inter-médiateurs

La figure 6.18 illustre les interactions entre composants lors de l' ajout d'une connexion inter-médiateurs. On ne s'inquiète pas ici de la topologie de la fédération, seul un médiateur est impliqué. Dans un but de lisibilité, le diagramme ne modélise pas l'attente de l'acceptation par l'administrateur du médiateur 2 de la demande de connexion initiée par le médiateur 1.

La création de la connexion entre les médiateurs déclenche l'échange de leurs catalogues respectifs.

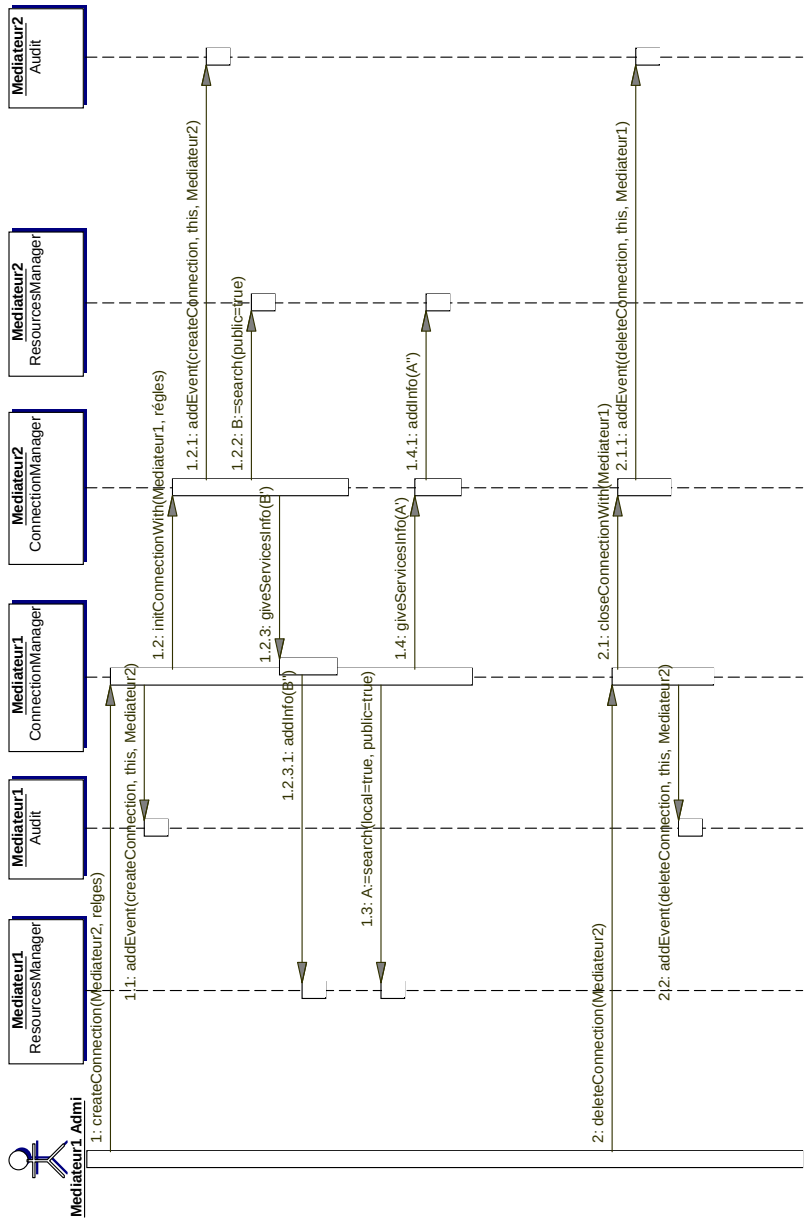


FIG. 6.18 – *Ajout d'une connexion inter-médiateurs*

6.6 Dynamique de la couche 1 du système

Cette section a pour but d'illustrer visuellement la dynamique des composants de la couche 1 du système au moyen de diagrammes de séquence. Nous reprenons entièrement la topologie de fédération donnée dans la section précédente.

6.6.1 Identification d'un utilisateur auprès du système

Le diagramme de séquence de la figure 6.19 modélise le processus d'identification de l'utilisateur auprès du système.

L'utilisateur appelle le *SecurityManager* en lui remettant son identifiant d'utilisateur et son mot de passe. Ensuite, le *SecurityManager* vérifie auprès du *UserManager* si l'utilisateur est un utilisateur valide (login et mot de passe correct) du système. Pour finir, le *SecurityManager* remet à l'utilisateur son *passe* (contenant son login, une date de péremption et une signature du contenu pour éviter son altération) pour sa session de connexion du système.

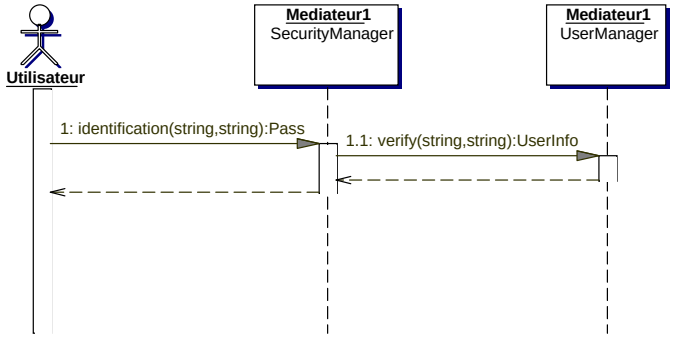


FIG. 6.19 – *Diagramme de séquence de l'identification d'un utilisateur*

6.6.2 Désidentification d'un utilisateur auprès du système

Le diagramme de séquence de la figure 6.20 modélise le processus de désidentification de l'utilisateur auprès du système.

Tout d'abord, l'utilisateur appelle le *SecurityManager* en lui remettant son *passe*. Ensuite, le *SecurityManager* le supprime de sa liste des *passes* en circulation.

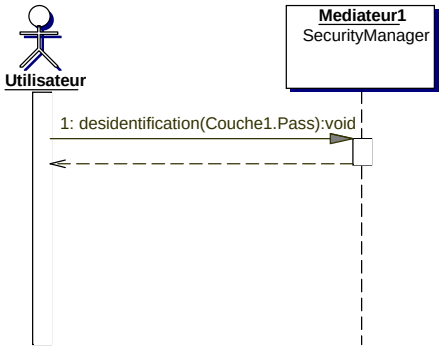


FIG. 6.20 – *Diagramme de séquence de la déconnexion d'un utilisateur*

6.6.3 Appel d'un service par un utilisateur

Le diagramme de séquence de la figure 6.21 modélise le processus d'appel de l'utilisateur à un service.

Premièrement, l'utilisateur effectue son appel en précisant le service qu'il désire utiliser, les paramètres d'entrée du service et son *passe* l'identifiant. Il recevra un identifiant du *job* dont il a provoqué l'exécution.

Deuxièmement, le *CallManager* vérifie auprès du *SecurityManager* si l'utilisateur bénéficie bien des droits pour le service qu'il désire utiliser. Le *SecurityManager* vérifie auprès du *UserManager* la validité de l'utilisateur dans le système.

Troisièmement, le *CallManager* va ensuite demander au *ProxiesManager* de lui fournir un *proxy* du service qu'il a demandé. Le *ProxiesManager* va, dès lors, créer un *proxy* et/ou la *route de proxies* menant au service réel.

Quatrièmement, le *CallManager* va enregistrer auprès du composant *AuditManager* la demande d'utilisation de service. Il va également enregistrer l'identifiant de *job* qu'il a remis à l'utilisateur.

Cinquièmement, lorsque l'exécution du service est terminée, le *CallManager* va déclencher la destruction de la *route* menant au service réel. Il va également inscrire auprès du composant *AuditManager* la durée de l'exécution ainsi que le coût de la *route* qu'il doit éventuellement payer au premier médiateur par lequel il a dû passer pour accéder à la *route* menant au service.

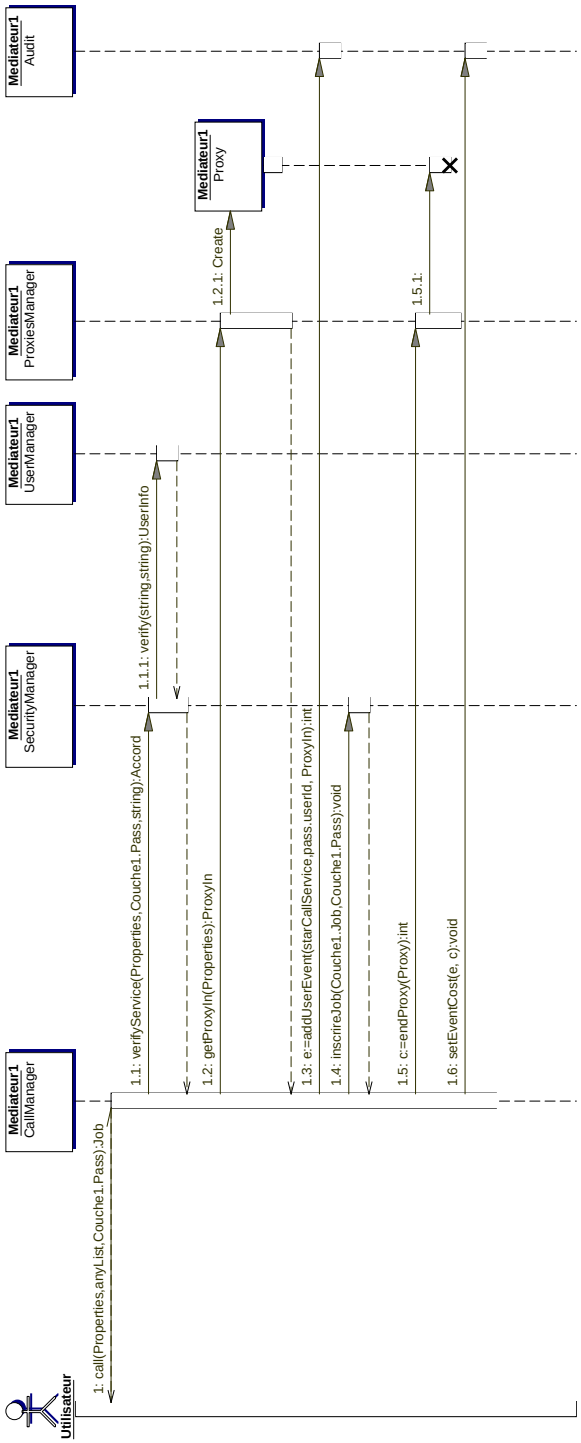


FIG. 6.21 – *Diagramme de séquence de l'appel d'un service*

6.6.4 Recherche par un utilisateur d'un service en catalogue

Le diagramme de séquence de la figure 6.22 modélise le processus de recherche de service(s) par l'utilisateur.

L'utilisateur s'adresse au *CallManager* en lui fournissant les critères de recherche qu'il a choisis afin d'obtenir une liste de services candidats disponibles au sein de la fédération. Ensuite, le *CallManager* s'adresse au *ResourcesManager* pour obtenir la liste des services correspondants aux critères de l'utilisateur et, sur base de la liste que le *ResourcesManager* lui retourne, le *CallManager* va vérifier auprès du *SecurityManager* les services pour lesquels l'utilisateur dispose des droits nécessaires. Le *SecurityManager* va filtrer cette liste (en ayant bien sûr contacté le *UserManager* afin de vérifier la validité de l'utilisateur) en fonction des droits dont dispose l'utilisateur. Pour finir, le *CallManager* va fournir cette liste filtrée à l'utilisateur.

6.6.5 Obtention par un utilisateur du résultat de l'exécution d'un service

Le diagramme de séquence de la figure 6.23 modélise le processus de récupération du résultat de l'exécution d'un service par l'utilisateur.

Tout d'abord, l'utilisateur contacte le *CallManager* en lui fournissant l'identifiant du job qu'il a reçu lorsqu'il a demandé l'exécution du service. Il fournit également son *passe* afin d'être identifié. Ensuite, le *CallManager* vérifie les droits sur le *job* et l'identité de l'utilisateur auprès du *SecurityManager*. Bien entendu, cette action sera enregistrée auprès du composant *AuditManager*. Pour terminer, le résultat, s'il est disponible, sera fourni à l'utilisateur.

6.6.6 Vérification par un utilisateur de l'état d'un *Job*

Le diagramme de séquence de la figure 6.24 modélise le processus de consultation de l'état d'exécution d'un service par l'utilisateur.

Premièrement, l'utilisateur contacte le *CallManager* en lui fournissant l'identifiant du *job* qu'il a reçu lorsqu'il a demandé l'exécution du service. Il fournit également son *passe* afin d'être identifié. Deuxièmement, le *CallManager* vérifie les droits sur le *job* et l'identité de l'utilisateur auprès du *SecurityManager*. Bien entendu, cette action sera enregistrée auprès du composant *AuditManager*.

Finalement, l'état d'avancement du *job* sera fourni à l'utilisateur.

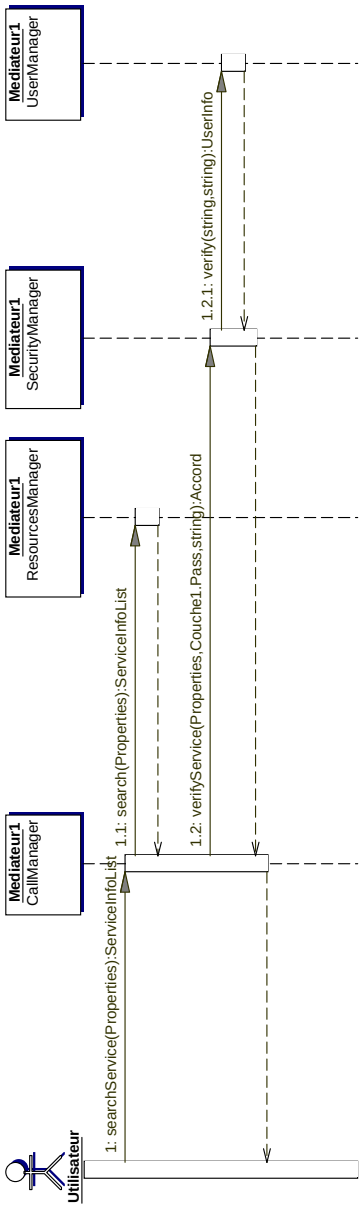


FIG. 6.22 – *Diagramme de séquence de la recherche en catalogue d'un(des) service(s)*

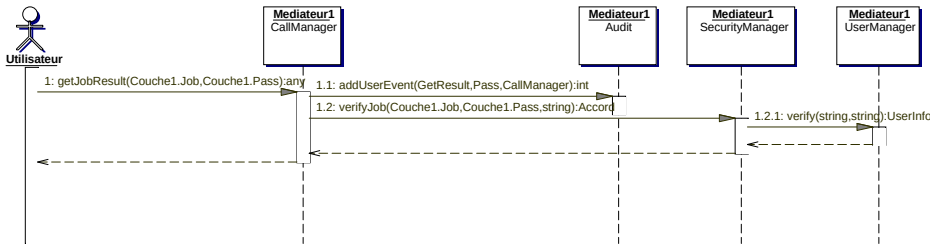


FIG. 6.23 – *Diagramme de séquence de l'obtention du résultat de l'exécution d'un service*

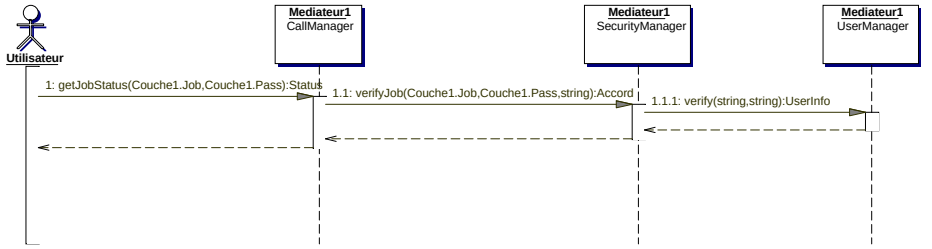


FIG. 6.24 – *Diagramme de séquence de la verification de l'état d'un Job*

6.7 Conception logique du système de gestion de la persistance

Description du schéma conceptuel global des données persistantes (voir figure 6.25 page 126):

- Un utilisateur est identifié par un login unique au sein d'un médiateur. Il est caractérisé par un nom, un prénom, un mot de passe et par des informations permettant de le contacter (adresses, numéro de téléphone, etc.).
- Un utilisateur peut être supervisé par un autre utilisateur. Un utilisateur qui supervise un autre utilisateur doit appartenir au même site que le supervisé ou à un site contenant le site du supervisé (voir analyse des exigences page 72).
- Un utilisateur peut être le contact de zéro ou plusieurs sites (voir description de l'entité site). L'utilisateur qui est le contact d'un site doit y être attaché.
- Un utilisateur est attaché à un ou plusieurs sites.
- Un utilisateur a obligatoirement un profil.
- Un profil est identifié par un nom et caractérisé par une description. Un profil peut être lié à zéro ou plusieurs droits sur des services.
- Un utilisateur appartient à zéro ou plusieurs groupes d'utilisateurs.
- Un groupe est identifié par un nom et caractérisé par une description.
- Un groupe a zéro ou plusieurs droits sur des services.
- Un utilisateur peut avoir zéro ou plusieurs droits sur des services.
- Un utilisateur peut donner zéro ou plusieurs droits.
- Il existe quatre types de profils: "normal", "avancé", "expert", "bioinformaticien" et "administrateur".
- Un utilisateur peut valider un droit qu'un utilisateur a donné, si et seulement si, il a le profil "administrateur".
- Un utilisateur doit être le propriétaire d'un service ou être un utilisateur de profil administrateur pour pouvoir donner un droit.
- Un utilisateur possède zéro ou plusieurs services.
- Un droit porte sur un ou plusieurs services.
- Un site[9] est identifié par un nom et est caractérisé par une localisation. Un site peut faire partie intégrante d'un autre site (on a ainsi une relation d'inclusion d'un site dans un site. Par exemple, l'IBMM est inclus dans ULB qui est inclus dans EMBNET, etc.).

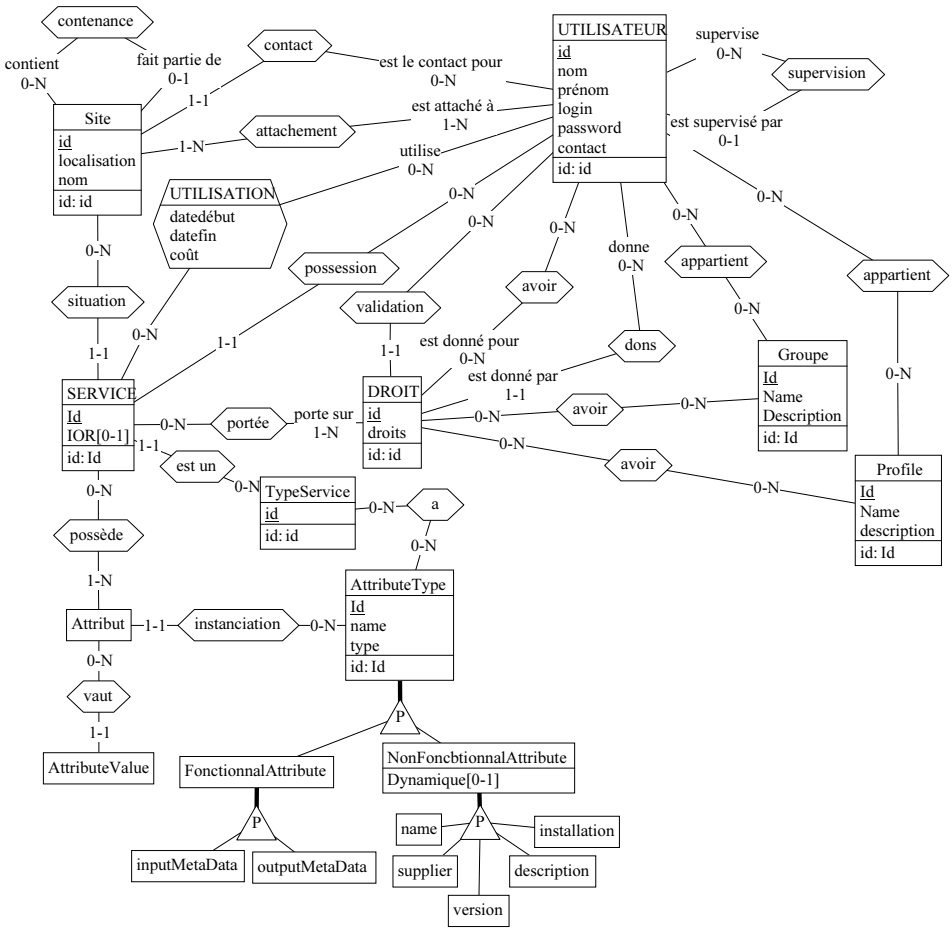
9. La notion de site est issue des exigences (voir page 72) et est une notion conceptuelle. Il est entendu que dans le cadre d'un système de persistance central à tous les médiateurs de la fédération, la notion de site serait une table de la base de données. Dans le cas ou chaque médiateur possède sa propre base de données, un site serait matérialisé par la base de données elle-même.

– Un site possède un et un seul responsable de contact. De même, un site se voit attaché un ou plusieurs utilisateurs.
– Au sein d'un site, sont présents zéro ou plusieurs services.

Description du schéma conceptuel des données persistantes des services (voir figure 6.26 page 127):

– Un service est situé sur un et un seul site.
– Un service appartient à un et un seul utilisateur.
– Un service est la portée de zéro ou plusieurs droits.
– Un service est identifié par un identifiant. Un service est caractérisé par un nom, une installation, une version, un "supplier", une description, un IOR ou non, un "status" et une règle d'exportation. Si un service a un IOR, alors ce service est local et on connaît son statut.
– Un service possède des attributs qui ont un type (AttributeType) et une valeur (AttributeValue).
– Un service a un Type de Service qui conditionne les types de ses attributs.
– Un type d'attribut est soit fonctionnel soit non fonctionnel.
– Un attribut fonctionnel est soit un attribut d'input ou bien un attribut d'output.
– Un attribut non fonctionnel est soit dynamique[10] ou non.

10. Un attribut dynamique est un attribut dont la valeur est modifiable à tout instant, sans intervention de l'administrateur.



Un UTILISATEUR doit être attaché à un SITE pour en être le contact.

L'UTILISATEUR superviseur d'un UTILISATEUR doit appartenir au même SITE que le supervisé ou à un SITE auquel appartient le SITE du supervisé.

Pour donner un droit sur un SERVICE, un utilisateur doit être son propriétaire ou avoir un profil d'administrateur.

Pour jouer un rôle dans les relations DEFINITION et VALIDATION, un utilisateur doit avoir un profil d'administrateur.

Pour tout sous-type de SERVICE, il existe un une entité de TYPE DE SERVICE le décrivant.

FIG. 6.25 – *Schéma conceptuel global des données persistantes*

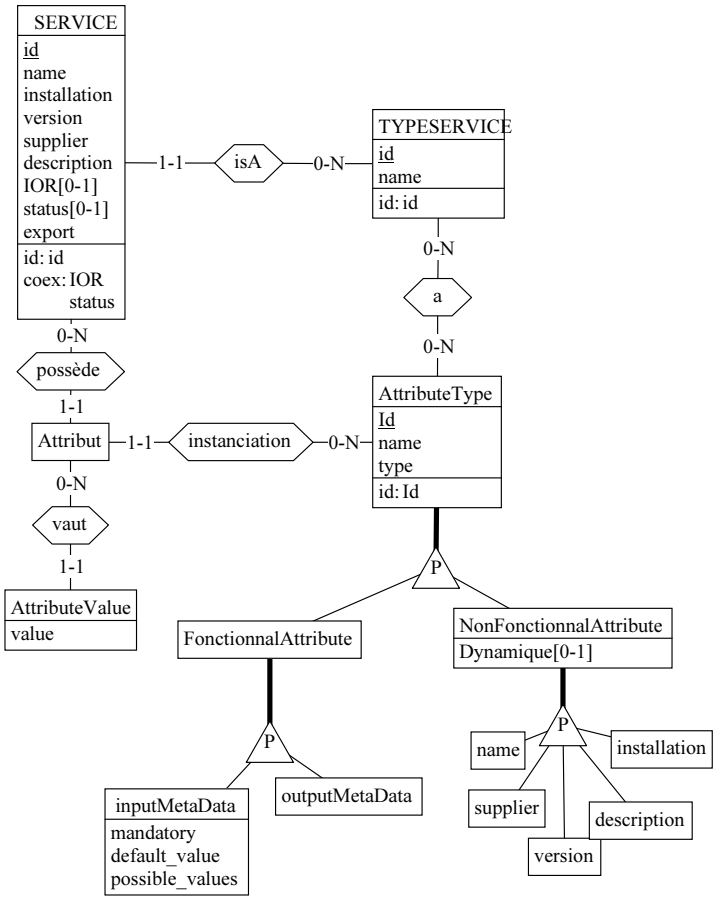


FIG. 6.26 – *Schéma conceptuel des données persistantes des services*

6.8 Interface homme-machine

Le but final du système est d'être beaucoup plus proche de l'utilisateur mais aussi de faciliter le développement d'outils à destination de celui-ci. En effet, dans l'état actuel des choses, l'utilisation d'un service nécessite de nombreuses adaptations de formats entre les arguments et les résultats de différents services. Il n'est pas rare que pour formater les arguments d'entrée d'un service, un preprocessing Perl soit nécessaire.

C'est pour cette raison que la fédération a pour but de fournir un outil de plus haut niveau évitant toutes ces manipulations écartant le biologiste ou le clinicien de sa tâche principale. Dans cette optique, il serait intéressant de créer des interfaces homme-machine (IHM) intelligentes et dynamiques s'adaptant de manière automatique au service à utiliser.

Dans le cadre de ce mémoire, nous ne nous sommes pas attardés sur ce point car notre priorité était de proposer une architecture avancée tenant compte des problèmes de montée en charge, de fiabilité et de compatibilité avec les contraintes liées à la distribution dans des environnements hétérogènes. Par ailleurs, nous devions aussi valider des principes qui avaient été proposés mais non prototypés dans le mémoire de nos prédécesseurs [DD03] (cf le *chaînage des proxies*).

Cependant, nous pouvons proposer une piste de réflexion pour une création dynamique d'IHM pour l'utilisation de services. Nous avons défini les services au moyen d'un schéma entité-association (figure 6.26). Ce schéma nous offre la possibilité de dériver une DTD XML de description des services. Sur base, de cette grammaire XML, nous proposons de concevoir un moteur d'IHM dynamique définissant les règles de transformation `XSLT` entre le mapping XML d'un service et une grammaire XML définissant de manière abstraite le format et le contenu d'IHM. Ainsi, il serait possible de générer automatiquement des interfaces HTML, Java Swing ou autres (Windows, QT3, GTK+, etc.) depuis leurs descriptions abstraites en XML [11].

Au delà de ce principe technique, limité aux IHM des services, la conception d'IHM se doit de passer par une phase d'analyse complète se basant essentiellement sur les cas d'utilisation décrits au cours du chapitre précédent. Ainsi, une découpe en tâches des scénarios d'utilisation correspondants aux cas d'utilisation et une définition des IHM liées à ces tâches devraient être faites. La réalisation de cette partie de l'analyse nous paraît être une condition essentielle pour l'adoption de ce système par les utilisateurs.

11. Pour la génération d'IHM à partir d'une description XML nous proposons de partir de systèmes existants tels que koalaGML [koa] ou XUL [Xul].

6.9 Conclusion

Au cours de ce chapitre, nous avons commencé par envisager, puis choisir, différents mécanismes nécessaires en vue de répondre aux exigences répertoriées dans le chapitre précédent. Ainsi, pour la recherche des services par les clients, nous avons décidé d'utiliser un système de requêtes basé sur un catalogue offert par chaque médiateur de la fédération. Ce catalogue, dont la diffusion au sein de la fédération est assurée par chaque médiateur, contient aussi des informations sur des services présents sur d'autres médiateurs.

Pour l'accès aux services, nous avons choisi de conserver le principe des *chaînes proxies* proposé par Laurent Debaisieux et Fernando Desouza[DD03]. Nous avons étendu ce principe au moyen de *routes* de *proxies* construites et élues en fonction de propriétés dynamiques fixées par les services et les médiateurs intermédiaires.

Ensuite, nous avons défini sommairement les mécanismes nécessaires à assurer la sécurité du système. D'un côté, la sécurité intra-médiateur est gérée selon un principe "classique" de gestion des droits d'utilisateurs locaux. Un médiateur autorise ou non les accès aux services de son catalogue sur base de règles locales définies par son administrateur. En ce qui concerne la sécurité inter-médiateurs, nous l'avons limitée à une gestion de la propagation des informations concernant les services (service local, service partagé avec un ou des voisins directement connectés ou avec l'ensemble de la fédération). Il est important de remarquer que ce mécanisme implique une importante confiance entre les administrateurs des médiateurs connectés.

Après avoir défini ces mécanismes, nous avons défini l'architecture logique du système. Pour ce faire, nous avons choisi une architecture en couches offrant des fonctionnalités de plus en plus évoluées. Chaque couche se base sur les couches de niveaux inférieurs. Ainsi, la couche 0 assure les fonctionnalités de base du système telles que l'accès aux services avec la création de *proxies* et le choix des *routes*, la maintenance du catalogue de services local au médiateur et la gestion de connexions inter-médiateurs. La couche 1 fournit les fonctionnalités nécessaires aux principales interactions entre clients et médiateurs (la gestion des utilisateurs, de leurs droits, de leurs appels de services et de la sécurisation de leurs échanges). Les couches supérieures, quant à elles, n'ont pas été définies. Cependant, nous avons déjà isolé dans la couche 2 les fonctionnalités liées aux utilisateurs avancés ou experts, telle que la réalisation de scripts d'analyses complexes. Enfin la couche 3 devrait contenir les composants nécessaires à la création d'un système expert ou d'aide à la décision apportant des fonctions à forte valeur ajoutée à l'ensemble du système.

Nous avons ensuite défini les interactions entre les différents composants des couches par des diagrammes de séquence. A cette occasion nous avons réfléchi sur le mode de communication entre composants de médiateurs différents. Nous avons décidé, dans un premier temps, d'autoriser les connexions

directes entre de tels composants, quitte à apporter plus tard un système de communication plus complexe mais plus souple en terme de déploiement.

Pour terminer notre analyse logique, nous avons réalisé un schéma de persistance de l'ensemble des données du système. Il est important de remarquer que nous nous sommes focalisés sur la partie fonctionnelle du projet, laissant de côté la question des interfaces homme-machine.

Chapitre 7

Conception Physique

7.1 Introduction

L'architecture physique précise ou agrège les composants isolés dans l'architecture logique. L'agrégation ou la décomposition de composants est réalisée selon leur distribution. Un composant physique est un composant indépendant des autres composants physiques dans le sens où il se situe dans un environnement d'exécution distinct. Si un composant logique doit être exécuté dans un espace qui lui est propre, il est conservé comme composant physique. Un ensemble de composants logiques fortement liés pouvant être exécuté comme un tout est regroupé dans un seul composant physique. Un composant logique peut aussi être décomposé en composants physiques plus petits pour, par exemple, assurer une meilleure robustesse au système.

La *Conception Logique* nous a permis de définir une architecture logique pour le système à mettre en place. Cependant, telle quelle, cette architecture n'est pas directement utilisable pour implémenter le système. Dans cette partie de *Conception Physique*, nous allons définir plus précisément les éléments du système afin de préparer leur implémentation. Pour ce faire, nous allons d'abord définir l'architecture physique des composants du système ainsi que l'architecture physique de persistance.

A partir de cette phase de conception, la solution proposée cesse d'être générique et dépend de choix physiques. En raison des contraintes liées au temps de réalisation de l'analyse, nous n'avons spécifié que les parties du système que nous comptions réellement implémenter, soit la majeure partie des couches 0 et 1.

7.2 Distribution des composants

L'architecture logique définit les composants du système selon leurs fonctionnalités. Nous avons vu qu'un système distribué se découpait en composants distribués, relativement indépendants les uns des autres. Il est possible

de considérer chaque composant logique comme un composant physiquement distribué, mais, souvent, il est préférable d'agréger ou de raffiner en scindant certains de ces composants logiques en plusieurs composants physiques.

Pour l'architecture physique, nous avons conservé une structure de composants séparés en couches. D'un point de vue distribution, chaque couche doit rester indépendante des autres, aucune agrégation de composants appartenant à plusieurs couches ne peut donc être faite. Les communications entre couche auront donc lieu via le *middleware* d'application distribuée choisi.

La figure 7.1 reprend les composants de l'architecture logique en précisant leur distribution physique.

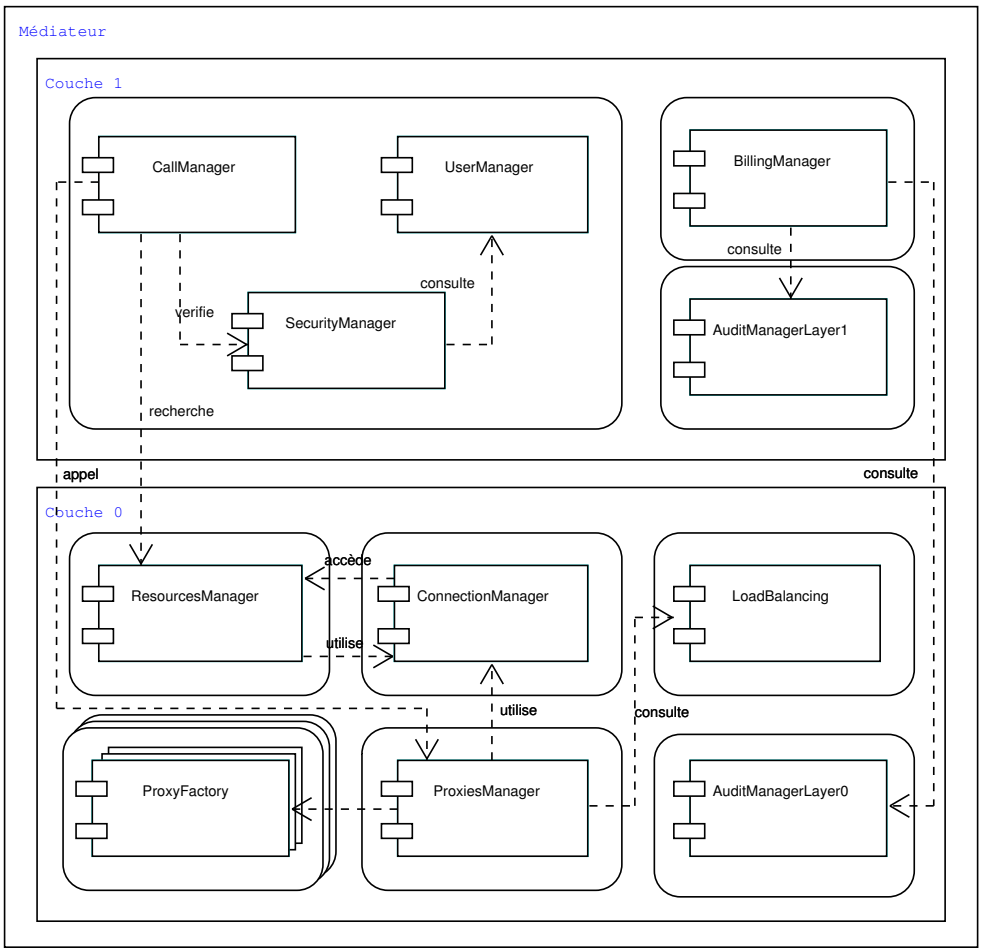


FIG. 7.1 – *Schéma de distribution des composants*

Les fonctionnalités remplies par chaque composant étant suffisamment distinctes et importantes, nous n'avons pas envisagé d'agréger les compo-

sants de la couche 0. Ainsi, on retrouve au niveau physique essentiellement les mêmes composants qu'au niveau logique: le *ConnectionManager*, le *ResourcesManager*, le *LoadBalancing*, l'*AuditManager* et le *ProxiesManager*. Du point de vue de sa distribution physique, chaque composant logique de la couche 0 devient un composant distribué du système et peut ainsi être déployé sur un hôte distinct de celui des autres composants de la couche 0.

Cependant, le cas du *ProxiesManager* est particulier. Son rôle consiste à gérer le cycle de vie des *proxies* des services bioinformatiques. Comme chaque utilisation de service au sein de la fédération donne lieu à la création au sein de chaque *ProxiesManager* d'un nouveau *proxy* dans chacun des médiateurs présents sur la *route* menant au service, il nous est apparu nécessaire de raffiner la distribution de ce composant.

Nous l'avons donc raffiné selon les principes du design pattern de *Fabrique d'objets* (Abstract Factory) [GHJV94] tout en conservant un composant physique *ProxiesManager* qui agit par rapport aux autres composants comme sa contrepartie logique. Il assure ainsi le rôle de *Factory Finder* pour trouver, à chaque fois que nécessaire, la *fabrique* de *proxies* (*Proxy Factory*) la plus adaptée selon la situation et le moment. Les `Proxy Factories` assurent la tâche de gestion des *proxies* et peuvent être distribuées sur des hôtes différents ou ajoutées en fonction de la montée en charge (nombre de *routes* à construire). Le *ProxiesManager* sert ainsi d'intermédiaire entre les autres composants et les *Proxy Factories*, tout en répartissant de manière optimale leur charge de travail.

Pour la couche 1, nous avons regroupé trois des composants logiques en un seul composant physique. Ainsi, les composants *UserManager*, *SecurityManager* et *CallManager* ont été regroupés ensemble. En effet, un médiateur ne doit servir qu'un nombre restreint de clients; et c'est pour cette raison qu'il ne nous a pas paru pertinent de distribuer les composants intervenant dans leurs interactions. Le composant *BillingManager*, quant à lui, agit de façon plus indépendante et n'est pas nécessaire au fonctionnement du système, il peut donc constituer un composant physique à lui seul.

Dans les deux couches (0 et 1), les composants *AuditManager* sont distincts des autres composants. Selon le cas, un médiateur peut très bien utiliser la même instance de ce composant pour assurer son rôle dans les deux couches.

7.3 Schéma de persistance par composant distribué

Lors de la conception logique, nous avons défini la nature et la structure de l'ensemble des données persistantes du système. Physiquement, les données sont conservées par l'un ou l'autre, voire plusieurs composants du système. Les schémas suivants présentent les données dont certains compo-

sants sont responsables.

7.3.1 Schéma de persistance du *UserManager* et du *SecurityManager*

La gestion des données persistantes des composants *SecurityManager* et *UserManager* que présentées sur la figure 7.2 sera assurée au moyen d'un SGBD de façon à obtenir des recherches rapides et efficaces.

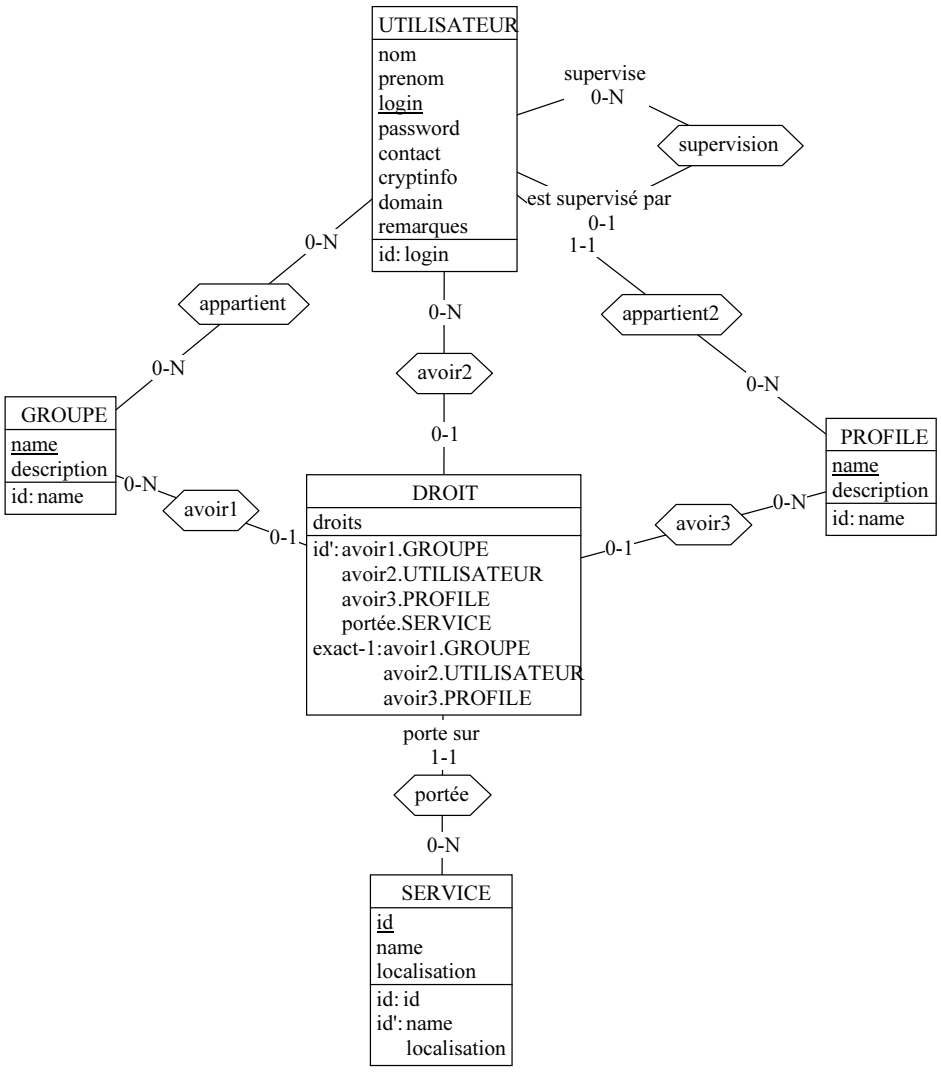


FIG. 7.2 – *Schéma de persistance des composants* SecurityManager *et* UserManager

7.3.2 Schéma de persistance du *ResourcesManager*

La gestion des données persistantes du composant *ResourcesManager* présentées sur la figure 7.3 sera assurée au moyen d'un SGBD de façon à obtenir des recherches rapides et efficaces.

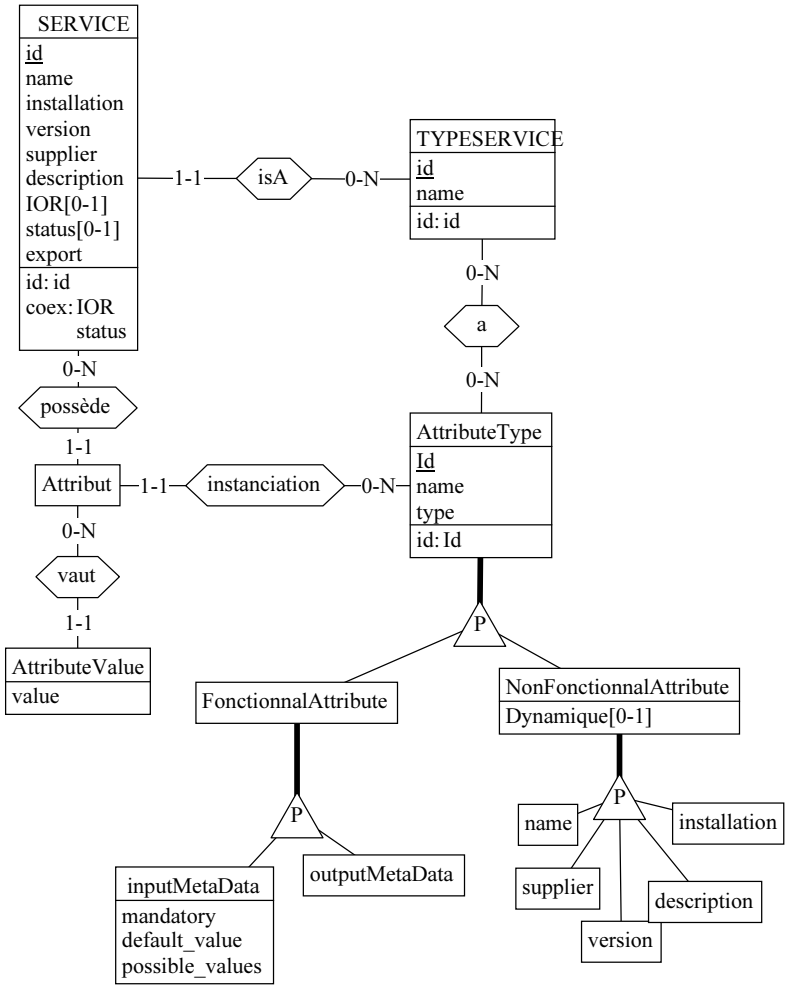


FIG. 7.3 – *Schéma de persistance du composant* ResourcesManager

7.3.3 Schéma de persistance de l'*AuditManager*

La gestion des données persistantes du composant *AuditManager* présentées sur la figure 7.4 sera assurée en "interne" par le composant et les événements seront enregistrés sous un format XML.

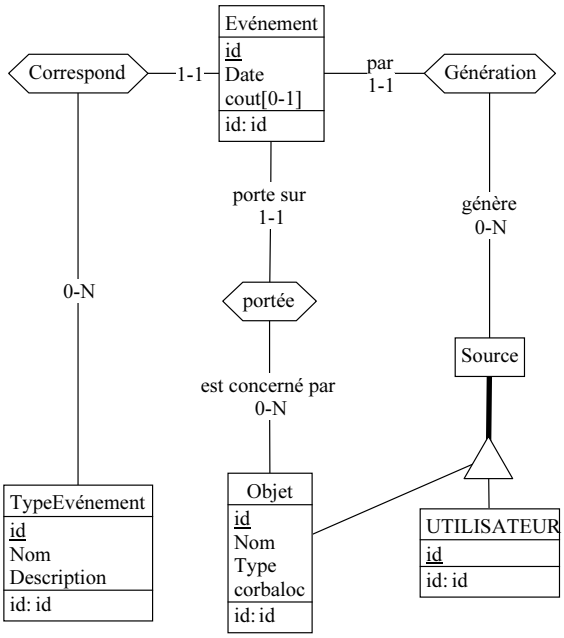


FIG. 7.4 – *Schéma de persistance du composant* AuditManager

7.4 Conclusion

Durant cette phase de conception physique nous avons défini l'architecture physique de l'application en agrégeant ou en raffinant les composants logiques issus de la phase précédente. Dans la couche 0, on retrouve globalement les mêmes composants à l'exception du *ProxiesManager* qui a été raffiné selon les principes du design pattern de *Fabrique d'objets*. Les composants logiques de la couche 1 ont été regroupés en un seul composant physique à l'exception du *BillingManager*. Nous avons également raffiné le schéma de persistance global du système afin de préciser la localisation des données au sein de chaque composant physique des médiateurs et les systèmes de gestion de la persistance à utiliser.

Chapitre 8

Implémentation

8.1 Introduction

Dans le cadre de notre stage de quatre mois à l'unité de bioinformatique de l'Université Libre de Bruxelles (ULB), nous avons réalisé un prototype de l'architecture proposée dans notre analyse. L'implémentation de ce prototype s'est déroulée du 28 Octobre 2002 au 20 Décembre 2002.

L'objectif de ce prototype était multiple. D'une part présenter "quelque chose" de concret à nos "clients" et leur montrer la faisabilité d'un tel système grâce à un prototype. D'autre part, ce prototype devait nous permettre de valider un ensemble de principes énoncés dans le mémoire [DD03] et qui n'avaient pas été prototypés (les *chaînes de proxies*, la gestion des connexions entre médiateurs). Pour finir, il devait nous permettre de valider et critiquer les principes et mécanismes que nous avons mis en avant dans notre conception logique:

- Le *chaînage des proxies* grâce aux *routes de proxies* remplaçant les *chaînes de proxies*.
- La création, le choix et la destruction dynamique des *routes de proxies*.
- L'architecture en couche proprement dite.
- Les principes de connexions entre médiateurs.
- Le protocole d'échange des descriptions de services dans la fédération et la synchronisation du catalogue des services auprès des médiateurs de la fédération.
- La gestion des exécutions asynchrones permettant à un utilisateur de lancer une exécution et de récupérer plus tard son résultat.

Lors de cette implémentation, nous avons tenté de respecter au maximum notre découpe en composants et les schémas de données proposés dans la conception logique et la conception physique.

8.2 Eléments de l'architecture implémentés

En regard du temps dont nous disposions mais aussi des éléments que nous avions spécifiés, il ne nous était pas possible d'implémenter toute notre architecture. Nous avons dès lors choisi les éléments les plus importants à prototyper et nous avons laissé de côté la couche 2 (langage de script et ses outils) et la couche 3 (SIAD et système expert). Ces éléments de l'architecture devront être approfondis et spécifiés de manière plus complète dans d'autres travaux car ils requièrent une étude complète des besoins et un fonctionnement parfait des mécanismes "de base" offerts par les couches inférieures. Dans cette optique, nous avons également implémenté des mécanismes simples pour la gestion de la sécurité et de l'audit.

Ainsi, nous nous sommes exclusivement consacrés aux éléments de la couche 0 et 1 que nous devions valider. Nous avons implémenté complètement un *ResourcesManager*, un *ConnectionManager*, un *ProxiesManager* et un *CallManager*. Par contre nous nous sommes limités à une implémentation partielle et simpliste des composants *SecurityManager*, *UserManager* et *AuditManager*. Ces composants devront être raffinés ultérieurement de manière plus approfondie.

Pour effectuer nos tests, nous avons également implémenté un service bioinformatique "vide" ayant pour but de permettre le test de la diffusion du catalogue des services au sein de la fédération, la recherche de services, la recherche d'une *route* mais aussi la *création* d'une *route* menant à un service. Richard Kamuzinzi (informaticien de l'IBMM), quant à lui, nous a implémenté un service bioinformatique respectant la norme BSA. Il offre la possibilité de tester une exécution réelle d'un service bioinformatique et permet aussi la génération de données sur base de paramètres tels que le temps de traitement et la taille des données à générer.

De notre côté, nous avons implémenté un client léger permettant l'exécution d'appels au service implémenté par Richard Kamuzinzi, la recherche de services dans le catalogue du médiateur, la consultation de l'état d'une exécution d'un service et la récupération de son résultat. Notre mémoire ne portant pas sur les IHM, nous n'y avons pas apporté de soin particulier.

8.3 Choix d'implémentation

8.3.1 Le *FrontEndClient*

Comme nous l'avons déjà précisé lors de la conception logique, la couche 1 joue le rôle d'un `proxy` entre le client et la couche 0. C'est pour cette raison que nous avons pris la décision de distribuer la couche 1 comme un seul composant (voir section 7.2 page 131 et figure 8.1 page 140) vis-à-vis du client. L'accès au médiateur par celui-ci, pour des raisons de facilité et de masquage des détails d'implémentation de la couche 1, doit se faire par un

composant "façade" unique. En effet, le client est susceptible de s'adresser à trois composants de la couche 1: le *SecurityManager*, le *CallManager* et l'*AuditManager*. Nous avons dès lors défini l'interface IDL du composant FrontEndClient. Ce composant joue le rôle de `proxy` entre le client et les composants implémentant la couche 1.

Le code IDL du FrontEndClient:

```
#pragma prefix "federgen"

#include <mediator.idl>

module corba
{
    struct JobTicket
    {
        long id;
    };

    struct Pass
    {
        string userLogin;
        string timeStamp;
        string passSignature;
    };

    exception CallErrorException{};
    exception BadLoginPasswordException{};
    exception AlreadyLoggedException{};
    exception NoLogoutException{};
    exception BadPassException{};
    exception BadJobTicketException{};
    exception JobNotFinished{};

    interface UserFrontEnd
    {
         Operation permettant d effectuer un appel sur un service

        JobTicket call(in federgen::mediator::Node searchtree,
            in CosPropertyService::Properties param,in Pass pass)
            raises (CallErrorException);

         Operation permettant d obtenir une liste de services
           correspondant a certains criteres

        corba::ServiceDescriptionList searchService(
            in federgen::mediator::Node searchtree,in Pass pass);

         Operation permettant de connaitre le statut d execution
           d un service

        DsLSRAnalysis::AnalysisState getJobStatus(in JobTicket job,in Pass pass)
            raises (BadPassException,BadJobTicketException);
```

```
        Operation permettant d obtenir le resultat de l execution d un
          service

        CosPropertyService::Properties getJobResult(in JobTicket job,in Pass pass)
            raises (BadPassException,BadJobTicketException,JobNotFinished);

        Operation permettant de s identifier aupres du systeme

        Pass login(in string login,in string password)
            raises (BadLoginPasswordException,AlreadyLoggedException);

        Operation permettant de se desidentifier aupres du systeme

        void logout(in Pass pass) raises (NoLogoutException);
    };
};
```

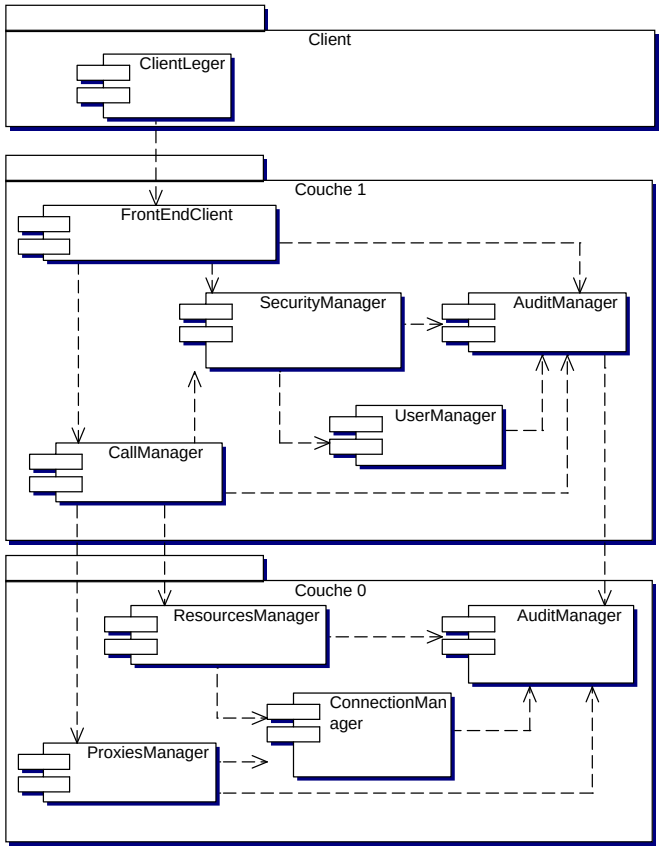


FIG. 8.1 – *Dépendances des composants d'implémentation*

8.3.2 Connexions entre médiateurs

Par simplicité et par manque de temps pour concevoir les IHM correspondantes, nous avons simplifié le principe des connexions bilatérales proposé dans la conception logique. Ainsi, lorsqu'un médiateur 1 demande à se connecter à un médiateur 2, la demande de connexion est automatiquement acceptée (voir figure 8.2).

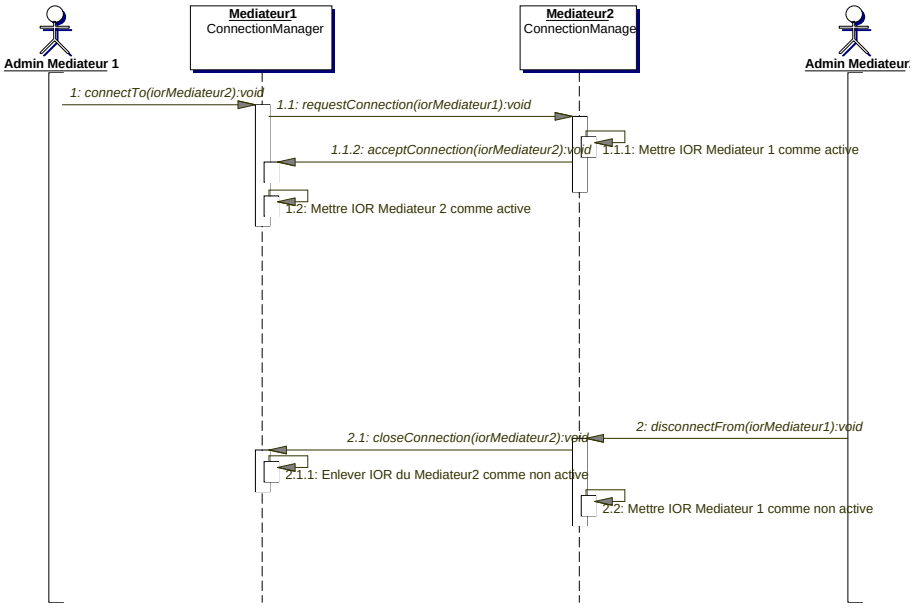


FIG. 8.2 – *Diagramme de séquence de la création et suppression de connexions entre médiateurs*

8.3.3 Composant *ProxiesManager*

Le *ProxiesManager*, comme annoncé dans la conception physique, est basé sur le design pattern de *Fabrique d'objets* et joue le rôle de *Factory Finder* permettant ainsi à un composant de s'adresser à lui pour obtenir la référence d'une *Factory*. Cette *Factory* se charge de "construire" l'instance d'un objet sur son hôte. L'utilisation de ce design pattern dans notre cas est intéressante car elle permet de déplacer la construction et l'exécution des *proxies* sur des hôtes différents, répartissant ainsi la charge de travail qu'ils suscitent.

Dans notre implémentation, afin de conserver notre découpe initiale en composants et les rôles identifiés, les mécanismes de sélection et d'appel aux *factories* sont directement gérés par le *ProxiesManager*. C'est pour cette

raison que nous avons modifié légèrement le principe du *Factory Finder*: le *CallManager* s'adresse au *ProxiesManager* pour obtenir non pas une référence vers une *Factory* de *proxies* mais directement la référence d'un *proxy*.

Les interfaces suivantes (voir figure 8.3) ont dès lors été définies:

- *ProxyFactory* se charge de la création de *proxies*
- *MotherProxyFactory* hérite de l'interface *ProxyFactory* et se charge en plus de gérer les inscriptions et désinscriptions de *Factories*.
- *ProxiesManager* hérite de l'interface *MotherProxyFactory* et se charge en plus d'offrir une méthode `getProxy()` permettant au *CallManger* d'obtenir un *proxy* vers le service qu'il désire.

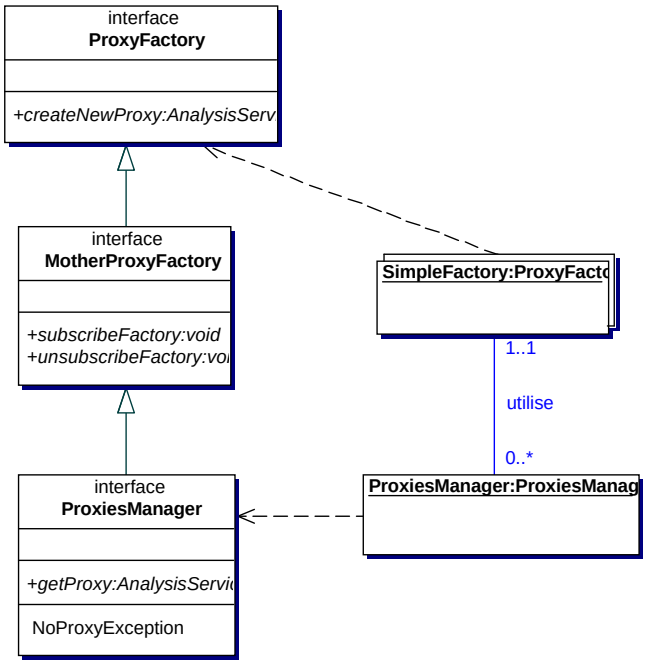


FIG. 8.3 – *Diagramme des interfaces des* `Factories` *de* proxies *et leurs classes d'implémentation*

Le code IDL des `Factory` de *proxies*:

```
#include <mediator.idl>

#pragma prefix "federgen"

module corba
{
    exception NoProxyException{};
```

```
    interface ProxyFactory
    {
         Operation de creation d un proxy

        DsLSRAnalysis::AnalysisService createNewProxy(
            in DsLSRAnalysis::AnalysisService proxy);
    };

    interface MotherProxyFactory: ProxyFactory
    {
         Operation d inscription d une factory

        void subscribeFactory(in ProxyFactory fact);

         Operation de desinscription d une factory

        void unsubscribeFactory(in ProxyFactory fact);
    };
    interface ProxiesManager: MotherProxyFactory
    {
         Operation permettant l obtention d un proxy vers un service

        DsLSRAnalysis::AnalysisService getProxy(
            in federgen::mediator::Node service2search) raises (NoProxyException);
    };
};
```

Le fonctionnement du *ProxiesManager* tel que décrit dans la figure 8.4 est le suivant: Les *Factories* de *proxies* 1 et 2 s'inscrivent auprès du *ProxiesManager*. Lorsque le *Call Manger* demande la création d'un *proxy*, le *ProxiesManager* se charge de répartir au mieux la charge entre les différentes *Factories* de *proxies* enregistrées. Cette répartition peut être réalisée de diverses manières (gigue, compteur parcourant une liste, méthodes mathématiques, etc.). Les *Factories* ont aussi la possibilité de se désinscrire du *ProxiesManager*.

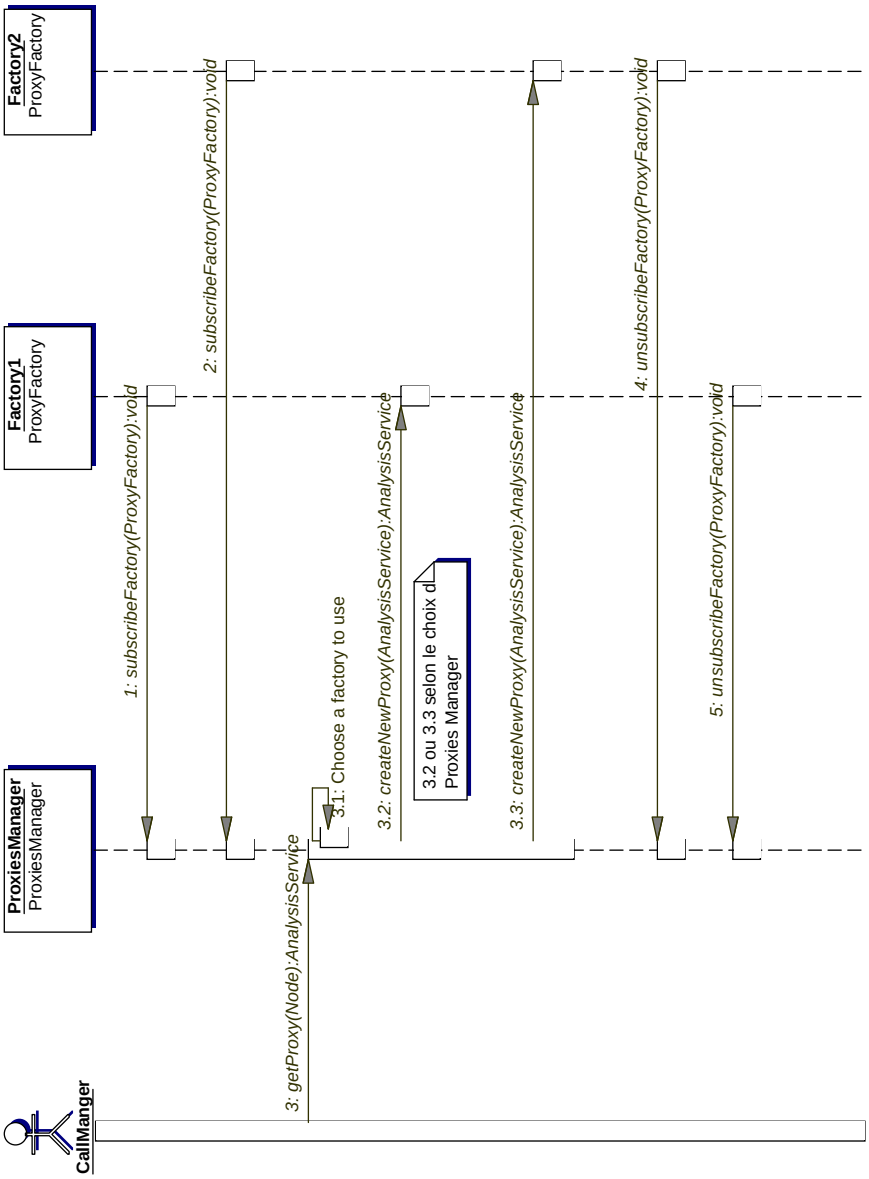


FIG. 8.4 – *Fonctionnement du* ProxiesManager

8.3.4 Approche par délégation des implémentations de composants

Pour implémenter nos composants CORBA distribués, nous avons opté pour une approche par délégation. Cette méthode permet d'implémenter les interfaces d'opérations générées par le compilateur IDL. Cette approche a un avantage par rapport à l'approche par héritage: elle permet l'implémentation de plusieurs interfaces simultanément au moyen d'une seule et même classe d'implémentation Java. Ceci n'est pas possible dans le cas d'une approche par héritage en Java (Java ne gère pas l'héritage multiple). De plus, cette méthode est obligatoire pour gérer en Java les éventuels héritages d'interfaces définis en IDL.

Dans l'exemple illustrer sur la figure 8.5, la classe d'implémentation d'un *proxy* implémente trois interfaces d'opérations:

- `org.omg.DsLSRAnalysis.AnalysisServiceOperations`
- `org.omg.DsLSRAnalysis.AnalysisInstanceOperations`
- `org.omg.DsLSRAnalysis.JobControlOperations`

Par souci de cohérence de l'ensemble du code, nous avons généralisé l'utilisation de cette approche par délégation à tous nos composants distribués.

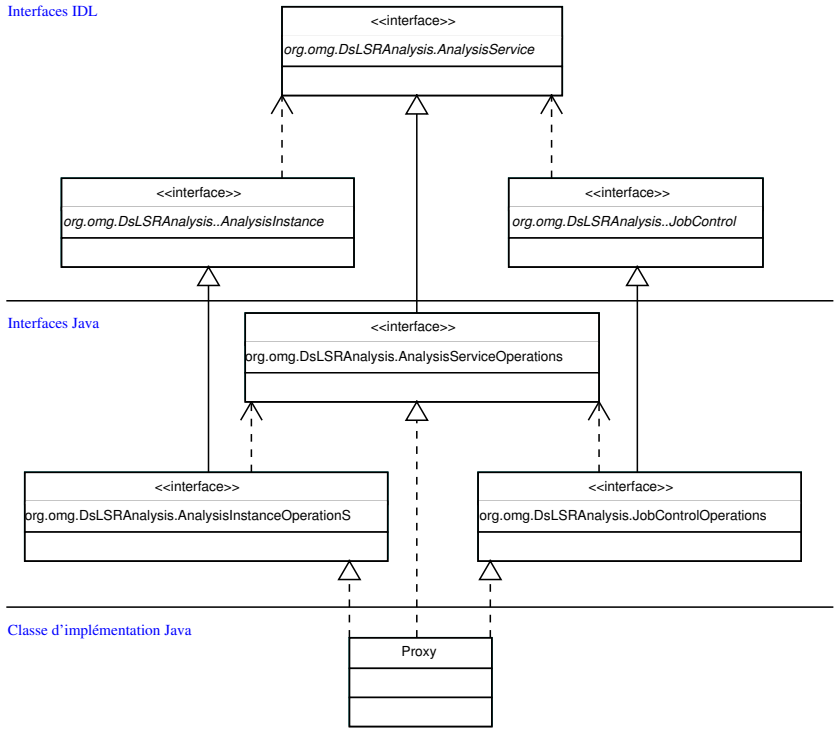


FIG. 8.5 – *Diagramme de classes de l'approche par délégation pour la classe d'implémentation* proxy

8.3.5 Langage de requêtes pour le catalogue des services

Pour l'implémentation des recherches dans le catalogue des services, nous avons développé un langage simple de contraintes basé sur les descriptions de services:

```
expression ::= expression opbin (expression) | contrainte
opbin ::= ET | OU
contrainte ::= String = String
```

Ce langage permet la constitution de requêtes plus ou moins complexes en *empilant* une série de disjonctions et/ou de conjonctions. Par exemple, la recherche d'un service portant le nom *Blast*, dont la version est la *2.24* et qui est hébergé soit à Namur, soit à Bruxelles est représentée par la contrainte `name = Blast ET (version = 2.4 ET (localisation = Namur OU localisation = Bruxelles))`.

D'un point de vue technique, pour représenter les contraintes, nous avons utilisé une structure d'arbre binaire. Les noeuds de l'arbre sont les égalités, les conjonctions et les disjonctions tandis que les feuilles sont les noms d'attributs ou leurs valeurs souhaités sous forme de chaînes de caractères. La figure 8.6 donne l'arbre binaire correspondant à l'exemple présenté ci-dessus. Pour la transmission des recherches, nous avons spécifié deux `struct` CORBA définissant la structure des arbres. Pour les opérations de recherches, un des argument est un arbre respectant la structure `struct Tree`.

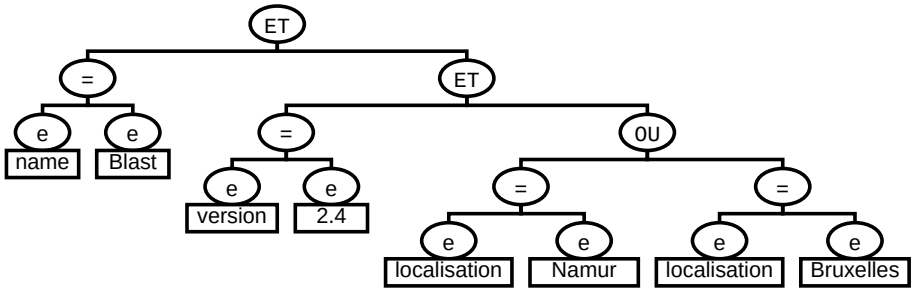


FIG. 8.6 – *Exemple d'arbre pour une recherche de services*

8.4 Outils utilisés

Cette section a pour objectif de présenter les outils que nous avons choisi d'utiliser lors de notre implémentation ainsi que les raisons qui ont motivé ces choix. Au regard de l'expérience que nous avons acquise lors de l'implémentation de notre prototype, nous critiquerons ces choix.

8.4.1 Sun Java2 Standard Edition 1.4.1

Pour implémenter notre prototype, nous avons choisi le langage Java. Ce choix a été motivé par le fait qu'en plus d'être gratuit, il s'agit du seul langage objet que nous connaissons et maîtrisons. De plus, opter pour Java comporte des avantages: ce langage respecte très bien le paradigme orienté objet, il permet d'écrire du code propre et lisible et il fournit un outil de génération de documentation automatique. Cet aspect est particulièrement intéressant pour la transmission, à d'éventuels successeurs, du code produit dans le cadre de notre mémoire. Par ailleurs, Java permet également un développement rapide d'IHM graphiques, ce qui dans une phase de prototypage constitue un atout.

Globalement, l'utilisation de Java nous a apporté entière satisfaction. Cependant, le seul point que nous pourrions éventuellement lui reprocher consiste en l'impossibilité de gérer complètement la durée de vie d'une instance d'un objet. En effet, en Java contrairement à C++, on crée un objet, on l'utilise et lorsqu'il n'est plus utilisé, on le "jette" en le déréférençant sans pour autant désallouer la mémoire qui lui a été attribuée. Le `garbage collector` de la `machine virtuelle Java` se charge de cette tâche. Bien que le système de `garbage collector` fonctionne plutôt bien dans la pratique, nous émettons quelques réserves lors de son utilisation dans le cadre d'un système distribué nécessitant la création fréquente de nombreux objets éphémères (La création de *routes* dans notre architecture constitue un parfait exemple).

8.4.2 Exolab OpenORB 1.3.0

Notre choix d'OpenORB, comme ORB Corba d'implémentation, a été motivé par le fait qu'OpenORB est Open Source et totalement gratuit. Il est également léger et portable car écrit en Java et nous avons l'habitude de l'utiliser dans le cadre de développements de systèmes distribués.

Bien qu'OpenORB soit un outil complet (il implémente Corba 2.4.2 complètement), performant et modulaire, son principal défaut est son inadaptation à une utilisation plus "professionnelle". En effet, la distribution d'OpenORB ne fournit pas, à l'instar d'autres ORB commerciaux, d'outils d'administration interactifs du bus. Ceux-ci offrent la possibilité de répliquer et de gérer la charge entre composants de manière plus interactive.

8.4.3 MySQL 3.23

Nous nous sommes portés vers ce SGBD Open Source car il est gratuit, léger et présent sur beaucoup de machines de développement et serveurs. De plus comme les autres outils, nous l'utilisons couramment. Par son aspect "léger", il convient parfaitement pour le la mise au point d'un prototype. Mais nous connaissions ses limites. En effet, MySQL n'est pas capable de

prendre en compte des requêtes SQL imbriquées dans une clause `WHERE`. Ainsi, MySQL sera incapable de "comprendre" la requête suivante: `select NCOM, DATECOM from COMMANDE where NCLI in (select NCLI from CLIENT where LOCALITE = 'Namur')` qui a pour objectif de sélectionner le numéro et la date des commandes passées par des clients de Namur [1]. MySQL ne gère pas non plus les contraintes d'intégrité standard SQL (excl, at-least-1, etc.). Malgré ces limitations, nous avons quand même décidé de l'utiliser quitte à gérer, nous-mêmes, dans notre code, les contraintes d'intégrité ou les requêtes imbriquées.

A posteriori, nous n'opterions plus pour MySQL. En effet, le défaut qui nous a le plus handicapés était, sans nul doute, le manque de gestion des contraintes d'intégrité élémentaires. Nous opterions, à l'avenir, pour un SGBD plus puissant tel que PostgreSQL ou Interbase de Borland. Par ailleurs, nous serions d'avis d'utiliser au maximum les possibilités offertes par les `triggers` et les `stored procedures`, de manière à réduire au maximum les connexions réseaux entre composants et le serveur du SGBD mais aussi pour obtenir une base de données intrinsèquement plus robuste.

8.4.4 CVS

Pour la gestion de notre *repository* du code et la synchronisation de nos différentes versions, nous avons utilisé le `Concurrent Versions System`; cet outil est très répandu et utilisé. Nous n'y étions pas familiarisé et cette implémentation était l'occasion de le découvrir.

CVS nous a donné entière satisfaction.

8.5 Conclusion

Suite à nos phases de conception, nous avons réalisé un prototype assurant la plupart des fonctionnalités du système que nous avions complètement spécifiées. Le but de ce prototype était de permettre le test et la validation des principes de fonctionnement choisis lors de la conception de l'architecture. Ce prototype a été réalisé en Java (JDK 1.4.1) en utilisant Exolab OpenORB 1.3.0 comme *midlleware* CORBA et MySQL 3.23 comme SGBD. Nous n'avons pas utilisé d'outils de développement spécifiques, si ce n'est le système de *repository* de code CVS.

1. Cet exemple est tiré du livre [Hai00]

Chapitre 9

Critiques

Ce chapitre a pour objectif de présenter une série d'autocritiques portant sur notre travail. Pour les réaliser, nous nous sommes basés sur notre expérience acquise lors de notre analyse, sur les problèmes que nous avons rencontrés lors de notre implémentation et sur les critiques émises par des tiers lors des présentations de nos spécifications et de notre prototype. En rapport à ces critiques, nous avons tenté de fournir de nouvelles pistes de réflexions exploitables dans une éventuelle poursuite du projet.

9.1 Invocations des services

Actuellement, pour accéder à un service, nous utilisons un système de *route de proxies* (voir section 6.1.2 page 85). Dans ce système, lors de chaque appel à un service, un *proxy* est créé par chacun des médiateurs situés sur la *route* entre le client et le service. La durée de vie d'une *route de proxies* est alors égale à la durée d'utilisation (le temps de traitement) du service invoqué. De plus, chaque message échangé entre le client et le service parcourt toute la *route de proxies*. Ce mécanisme est assez lourd car à un instant donné, la fédération compte autant de *proxies* que le nombre d'utilisations de services multiplié par la longueur moyenne des *routes*. Si ce système s'avère être élégant d'un point de vue conceptuel, il constitue, en pratique, une menace quant aux performances de l'ensemble de la fédération.

Notre système prévoit également que les *proxies* implémentent trois des interfaces du module d'analyse de la norme BSA (`AnalysisService`, `AnaysisInstance` et `JobControl`). L'utilisation de la norme BSA pour les interfaces des *proxies* est intéressante et simple, mais elle implique une certaine complexité d'implémentation qui n'est peut-être pas indispensable. En effet, elle impose pour chaque médiateur impliqué dans une *route*, l'utilisation de plusieurs objets distribués nécessaires à la transmission des messages. Ceci alourdit la gestion des *proxies* et augmente globalement la charge des médiateurs participant à la *chaîne de proxies*. Cependant, en pratique, seuls

les *proxies* présentés au *CallManager* doivent correspondre à la norme BSA et n'empêchent aucunement l'utilisation d'un mécanisme d'indirection plus léger entre ces *proxies* et les services auxquels ils offrent un accès.

9.2 Distribution du catalogue

Pour assurer la distribution du catalogue, nous avons envisagé plusieurs mécanismes. Finalement, nous avons retenu un système de réplication de l'information au sein de tous les médiateurs (voir section 6.1.1 page 79). Ce choix théorique a été effectué sur base des propriétés offertes par les différents mécanismes étudiés. Il prend également en compte certaines hypothèses telles que la stabilité des services. Il nous semble important, pour valider ou invalider le choix de la méthode de distribution du catalogue, d'effectuer des tests réels de performance, au moyen de simulations ou de prototypes.

En fonction du résultat de tels tests, si un système de réplication complet comme celui prévu initialement apparaît inadapté, il serait opportun d'envisager un système de recherches distribuées de type *query flooding* auquel serait adjoint un mécanisme de *caches agressifs*. Au moyen de telles caches dotés d'une taille et d'une durée de vie importantes, on obtiendrait, par *effet de bord* du mécanisme, une réplication quasi complète du catalogue.

9.3 Gestion de la montée en charge et facturation des services

Les mécanismes de gestion de la montée en charge des composants et de la gestion de la facturation des accès aux services ne nous semblent pas satisfaisants. Actuellement, cette *surveillance* est opérée au moyen des composants d'audit situés dans les couches 0 et 1 où tous les événements sont inscrits et conservés dans un journal accessible par les composants *LoadBalancing* et *BillingManager* leur permettant d'effectuer leurs traitements. Cependant, ce mécanisme rend difficile la gestion *temps réel* des événements car le système d'audit devrait être implémenté sur base d'un système de *publish and subscribe* [BMR+01] pour permettre une distribution rapide des événements.

Une alternative plus performante consisterait à utiliser un mécanisme basé sur le design pattern des *intercepteurs* [SSRB01] permettant l'interception des événements survenus au sein du système distribué. Cette utilisation des intercepteurs aurait comme avantage de ne pas imposer de *mémorisation* a priori des événements car ceux-ci ne seraient traités que si un composant le souhaite. Ce mécanisme deviendrait alors évolutif et tout événement pourrait être traité sans qu'aucun des traitements ne soit spécifié avant le mise en place du système (Ces traitements pourraient être spécifiés par la suite).

Ainsi, un système de facturation de l'utilisation des services pourrait être implémenté en quasi indépendance des autres fonctionnalités de la fédération.

9.4 Gestion des utilisateurs

Dans la fédération, la gestion des utilisateurs est uniquement réalisée par médiateur. Celui-ci ne connaît que ses utilisateurs locaux et ignore donc tout des utilisateurs des autres médiateurs. Les restrictions des accès aux services au niveau d'un utilisateur ne peuvent dès lors être opérées que par le médiateur auquel l'utilisateur appartient. Même si on définit un mécanisme limitant la propagation des informations des services et gérant les accès entre médiateurs, celui-ci n'offre que peu de flexibilité. Ainsi, il est impossible, par exemple, de limiter l'accès d'un service à un sous-groupe très spécifique d'utilisateurs répartis sur de nombreux médiateurs de la fédération .

C'est pour cette raison qu'un système de gestion globale, mais distribué, des utilisateurs de la fédération, offrirait une plus grande flexibilité dans la gestion des accès. Un tel système permettrait à la fédération de répondre de manière plus fine aux exigences des équipes de recherches internationales, pluridisciplinaires ou "pluri-institutionnelles". Sa mise en oeuvre permettrait aussi une facturation de l'utilisation des services plus aisée et de meilleure qualité par rapport à aujourd'hui où elle est limitée à une re-facturation *de proche en proche* entre médiateurs.

9.5 Sécurité du système

Bien que notre système de contrôle des accès aux différents services nous paraisse répondre aux exigences, la sécurité des médiateurs en tant que systèmes informatiques n'a pas été complètement étudiée. Ainsi, les communications, même sensibles, entre composants se font via le bus CORBA sans aucune précaution de sécurité. Il en est de même pour les communications entre clients et médiateurs. Dans la section 6.1.3 (90), nous émettions simplement l'hypothèse d'un cryptage de ces communications sans réellement l'explorer.

Par ailleurs, nous savons que le service de sécurité CORBA, permet la gestion de la sécurité des accès aux objets distribués du système et leur sécurisation. Ainsi, pour s'assurer de la sécurisation d'un futur déploiement de la fédération, il nous paraît pertinent de l'envisager au moyen de CORBA. Cependant, il serait également intéressant d'étudier les possibilités offertes par d'autres solutions indépendantes de CORBA qui permettraient une meilleure portabilité du système en cas d'utilisation d'autres *middlewares*.

9.6 Utilisation de la norme BSA

Malgré ses qualités, la norme BSA ne suscite pas l'unanimité au sein de la communauté des bioinformaticiens. Bien qu'elle se veuille indépendante de tout choix d'implémentation, y compris, pour certaines parties, de CORBA, elle repose sur plusieurs autres spécifications de l'OMG fort proches de CORBA et de sa *philosophie* de système distribué. Par ailleurs, l'utilisation de ces spécifications, dont la complexité de certains mécanismes visant la généricité, rend la norme BSA "trop lourde" par rapport aux besoins de la plupart des bioinformaticiens. Cette lourdeur est handicapante pour la réalisation rapide d'applications et décourage son emploi.

Cependant, un nouveau projet de norme est actuellement en préparation à l'OMG. Il devrait corriger les faiblesses de la norme BSA en revoyant et en ouvrant sa formalisation des données et en enrichissant la partie définissant le fonctionnement des outils d'analyses. Comme nous l'avons déjà précisé, si le module `BioObjects` de la norme est uniquement destiné au domaine des *sciences de la vie*, le module `AnalysisService` est quant à lui nettement plus générique et les mécanismes d'interactions client-service qu'il propose peuvent être utilisés en dehors du champ d'utilisation de la norme BSA. Il est plus que probable que la nouvelle norme en préparation conserve cette ligne de conduite. Une fois cette nouvelle norme acceptée, l'adaptation du système actuel ne devrait pas poser de réels problèmes car les mécanismes d'accès ne changent pas; et seul le changement des formats d'une partie des données demandera des modifications.

9.7 Recherche des services

Dans le système actuel, les recherches de services ont lieu localement dans les catalogues des médiateurs. Lors de la création des *routes de proxies*, le même système est utilisé pour vérifier l'existence d'un service local correspondant au service accessible par la *route de proxies*. Dans notre implémentation, les recherches se font uniquement par disjonctions et conjonctions de correspondances exactes entre les valeurs souhaitées et des valeurs effectives des attributs des services en catalogue. Ce principe ne permet pas, par exemple, de rechercher un service de `Nom` connu mais de `Version` au moins supérieur à une `Version` donnée ou dont le délai d'attente avant traitement ne dépasse pas la journée.

Une meilleure modélisation des requêtes nous semble nécessaire pour que le futur système puisse offrir un véritable langage de recherche dont les fonctionnalités soient en accord avec les exigences que nous avons identifiées.

9.8 Analyses des exigences

Lors de notre analyse des exigences, seuls les membres de l'unité de bioinformatique de l'IBMM ont été consultés. Leur profil correspond généralement à des scientifiques des *sciences de la vie* ayant également une très bonne *connaissance de terrain* de l'informatique. Cependant, il est important de préciser qu'ils ne représentent pas l'ensemble des utilisateurs potentiels du système. Ainsi, les exigences que nous avons dégagées l'ont été sur base d'interviews d'*acteurs bioinformaticiens* et *administrateurs* car nous n'avons pas pu consulter d'autres utilisateurs ayant une plus faible connaissance de l'outil informatique.

Pour ces raisons, il serait souhaitable de procéder à un raffinement des exigences auquel participeraient des *acteurs normaux*, *avancés* et *experts* du système.

9.9 Utilisation de CORBA

Pour l'ensemble des communications du système, nous avons choisi de n'utiliser que CORBA comme *middleware*. Cependant, comme nous l'avons relevé lors de la présentation de différents modes de communication (voir section 6.4 page 101), ce choix peut amener des difficultés. Il est important de remarquer que nous n'avons pas réellement envisagé et étudié de solutions alternatives à CORBA.

Cependant, il nous semble que, même s'il l'a influencée, ce choix n'a pas entraîné de répercutions trop importantes sur notre architecture. Ainsi, il devrait être possible de remplacer CORBA par une autre technologie pour assurer une partie ou l'ensemble des communications entre composants. Dans la section 6.4 (page 101), nous proposons d'ailleurs un système permettant d'utiliser n'importe quelle autre technologie pour les communications intermédiateurs (Java EJB, SOAP ou .NET) et n'imposant que quelques modifications de certains composants (*Proxies* et *ConnectionManager*).

9.10 Interfaces homme-machine

Comme nous l'avons déjà indiqué lors de la phase de conception logique (voir section 6.8 page 128), nous ne nous sommes pas attardés, dans ce travail, sur la conception d'interfaces homme-machine (IHM). Par conséquent, le prototype que nous avons proposé fournit des IHM simplistes ayant pour but de faciliter les tests et les présentations.

Cependant, la réalisation d'une véritable analyse pour la conception d'IHM est, selon nous, plus que souhaitable car elle conditionnera l'adoption du système par ses utilisateurs finaux.

Conclusion

La bioinformatique est une discipline en pleine évolution dont les besoins sont croissants. Actuellement, les outils informatiques de cette discipline sont encore fortement centralisés et peu compatibles entre eux. Pour pouvoir répondre à l'augmentation des besoins entraînée par son évolution et améliorer l'interopérabilité des ses outils, la bioinformatique doit s'adapter en passant d'un mode de fonctionnement centralisé à un mode de fonctionnement distribué.

Le projet FEDERGEN, initié par Marc Colet et Vincent Englebert et spécifié dans le mémoire de Laurent Debaisieux et Fernando Desouza, présentait une solution distribuée permettant le partage entre scientifiques de ressources bioinformatiques hétérogènes. Ce système proposait la création de *wrappers* englobant les outils existants et offrant une interface d'accès standardisée à ces outils par le respect de la norme "Biomolecular Sequence Analysis" (BSA) définie par l'OMG. La recherche et l'indexation des *wrappers* étaient assurées par des *médiateurs* organisés en une fédération partageant ces ressources. Ces médiateurs pouvaient également fournir des services à valeur ajoutée et permettre aux utilisateurs finaux du système d'accéder de manière transparente à l'ensemble des services disponibles.

Cependant, Laurent Debaisieux et Fernando Desouza n'avaient pas spécifié complètement tous les éléments indispensables à un système distribué opérationnel et certains des concepts qu'ils avaient avancés ne paraissaient pas optimaux ou complets en regard des exigences des utilisateurs visés.

Notre mémoire s'est inscrit dans le cadre de la poursuite de ce projet. Son objectif était de raffiner et de compléter la première approche du problème ébauchée par Laurent Debaisieux et Fernando Desouza.

Pour y parvenir, nous avons commencé une analyse complète du système incluant une phase d'analyse des exigences définissant les fonctionnalités attendues du système et une phase de conception logique définissant les bases de l'architecture du système et son fonctionnement. Nous avons ensuite adapté ces spécifications logiques lors de la phase de conception physique pour permettre l'implémentation de notre prototype.

L'architecture que nous avons développée au cours de notre analyse

conserve la plupart des idées de Laurent Debaisieux et Fernando Desouza et améliore certains principes. On peut y retrouver la fédération incarnée par un réseau de *médiateurs* inter-connectés auprès desquels sont inscrits les *wrappers*. Ceux-ci sont alors accessibles à l'ensemble des utilisateurs de la fédération.

Par rapport au travail initial de Laurent Debaisieux et Fernando Desouza nous avons:

- précisé et amélioré le mécanisme de *chaînes de proxies* au moyen de *routes* créées dynamiquement.
- défini un système de sélection de *routes* souple et dynamique permettant la répartition de la charge au sein de la fédération et offrant un meilleur contrôle de l'utilisation des *wrappers*.
- défini un mécanisme distribué de réplication du catalogue des services disponibles au sein de chaque *médiateur* de la fédération.
- précisé les modes de communication et d'interactions entre les *médiateurs*.
- précisé et simplifié les communications entre client et *médiateur*. Le *médiateur* n'est plus présenté à l'utilisateur selon la norme BSA et minimise les interactions nécessaires à la réalisation de ses tâches.
- défini une "architecture en couches" du *médiateur* et précisé les composants constituant celle-ci.
- développé un prototype de notre architecture.
- identifié les problèmes liés à notre architecture et entravant son déploiement en situation réelle.

Si le système que nous proposons n'est probablement pas adapté à une utilisation immédiate, nous pensons avoir, tout de même, fourni une base solide permettant le développement et l' implémentation d'une future fédération de partage de ressources bioinformatiques hétérogènes. En effet, dans le cadre d'une éventuelle continuation du projet, nous estimons que notre architecture "en couche" et sa découpe en composants peuvent être conservées. Néanmoins, certains choix techniques (communications, *middleware*, etc.) et mécanismes (*routes de proxies*, sécurité, etc.) devront être révisés dans le but d'améliorer l'adéquation du système aux contraintes physiques liées à son déploiement (topologie des réseaux de communication, etc.).

Bibliographie

[BGKS02] Daniel Brookshier, Darren Govoni, Navaneeth Krishnan, and Juan Carlos Soto. *JXTA: Java(TM) P2P Programming*. Sams Publishing, March 2002. ISBN 0-672-32366-4.

[Bio03a] BioCorba.org. Biocorba, 2003. `http://www.biocorba.org` (Dernière mise à jour : indéterminée) (Dernière visite : 17 décembre 2002).

[Bio03b] BioJava.org. Biojava, 2003. `http://www.biojava.org` (Dernière mise à jour : 12 décembre 2002) (Dernière visite : 24 mai 2003).

[Bio03c] BioPerl.org. Bioperl, 2003. `http://www.bioperl.org` (Dernière mise à jour : 15 janvier 2003) (Dernière visite : 24 mai 2003).

[Bio03d] BioPython.org. Biopython, 2003. `http://www.biopython.org` (Dernière mise à jour : 10 mai 2003) (Dernière visite : 24 mai 2003).

[BMR+01] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - A System Of Patterns*. Wiley, England, 2001. ISBN 0-471-95869-7.

[BSM03] BSML.org. Bioinformatic sequence markup langage, 2003. `http://www.bsml.org` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[CAA99] Jean-Michel Claverie, Stéphane Audic, and Chantal Abergel. La bioinformatique: une discipline stratégique pour l'analyse et la valorisation des génomes, 1999. `http://igs-server.cnrs-mrs.fr/jcnrs.html` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[CDKR02] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[Cona] World Wide Web Consortium. Extensible markup language (xml). `http://www.w3.org/XML/` (Dernière mise à jour : 15 mai 2003) (Dernière visite : 24 mai 2003).

[Conb] World Wide Web Consortium. Extensible stylesheet language transformations (xslt). http://www.w3.org/TR/xslt (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Conc] World Wide Web Consortium. Extensible stylesheet language (xsl). http://www.w3c.org/Style/XSL/ (Dernière mise à jour : 16 janvier 2003) (Dernière visite : 24 mai 2003).

[Cond] World Wide Web Consortium. Web services. http://www.w3.org/2002/ws/ (Dernière mise à jour : 14 mai 2003) (Dernière visite : 24 mai 2003).

[Cou02] Damian Counsell. Bioinformatics frequently asked questions, December 2002. http://bioinformatics.org/faq/ (Dernière mise à jour : 7 mai 2003) (Dernière visite : 24 mai 2003).

[Dan02] Jérôme Daniel. *Au coeur de CORBA avec Java*. Vuibert, Paris, France, seconde edition edition, 2002. ISBN 2-7117-8685-4.

[DD03] Laurent Debaisieux and Fernando Desouza. Partage de ressources bioinformatiques hétérogènes. Pour obtenir une copie contactez http://www.info.fundp.ac.be, January 2003.

[dT03] Agence Walonne des Télécommunications. Fiches de l'awt, 2003. http://www.awt.be/cgi/fic/fic_menu.asp (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Edw02] James Edwards. *Peer-to-Peer Programming on Groove®*. Addison Wesley, March 2002. ISBN 0-672-32332-X.

[Eng02] Vincent Englebert. Conception des systèmes d'information coopératifs. Namur, Belgique, http://www.info.fundp.ac.be/~ven/CIS/NotesDeCours/, 2002.

[Eng03] Vincent Englebert. Conception des systèmes d'information coopératifs matières approfondies. Namur, Belgique, http://www.info.fundp.ac.be/~ven/CISma/NotesDeCours/, 2003.

[fBI03] National Center for Biotechnology Information. National center for biotechnology information, 2003. http://www.ncbi.nlm.nih.gov (Dernière mise à jour : 7 mai 2003) (Dernière visite : 24 mai 2003).

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Pattern - Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, 1994. ISBN 0-201-63361-2.

[GJ01a] Cynthia Gibas and Per Jambeck. *Developing Bioinformatics Computer Skills*. O'Reilly, April 2001. ISBN 1-56592-664-1.

[GJ01b] Cynthia Gibas and Per Jambeck. *Introduction à la bioinformatique*. O'REILLY, Paris, France, 2001. ISBN 2-84177-144-X.

[Groa] Object Management Group. Biomolecular sequence analysis. `http://www.omg.org/technology/documents/formal/biomolecular_sequence.htm` (Dernière mise à jour : 16 mai 2002) (Dernière visite : 24 mai 2003).

[Grob] Object Management Group. Object management group. `http://www.omg.org/` (Dernière mise à jour : 15 mai 2003) (Dernière visite : 24 mai 2003).

[Hai00] Jean-Luc Hainaut. *Bases de données et modèles de calcul: Outils et méthodes pour l'utilisateur - Cours et exercices - 2ème Edition*. Dunod, 2000. ISBN 2 10 004553 9.

[Ins01] European Bioinformatics Institute. Applab, 2001. `http://industry.ebi.ac.uk/applab/` (Dernière mise à jour : 16 janvier 2001) (Dernière visite : 24 mai 2003).

[Ins03] European Bioinformatics Institute. European bioinformatics institute, 2003. `http://www.ebi.ac.uk/` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Jab] Sami Jaber. Dotnetguru - .net. `http://www.dotnetguru.org/articles/architecturedotnet.htm` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Jac02] Jean-Marie Jacquet. Techniques d'intelligence artificielle. Namur, Belgique, 2002.

[jGu] Sun Microsystems jGuru. Enterprise javabeans technology fundamentals. `http://developer.java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html` (Dernière mise à jour : 22 mars 2003) (Dernière visite : 24 mai 2003).

[koa] koalaGML. koalagml. `http://koalagml.sourceforge.net` (Dernière mise à jour : 10 janvier 2003) (Dernière visite : 24 mai 2003).

[LGG01] Nicholas M. Luscombe, Dov Greenbaum, and Mark Gerstein. What is bioinformatics? an introduction and overview, 2001. `http://bioinfo.mbb.yale.edu/~nick/bioinformatics/` (Dernière mise à jour : 10 octobre 2000) (Dernière visite : 24 mai 2003).

[Lok] Mohamed N. Lokbani. Aspect avancés en java - javabeans. `http://www.iro.umontreal.ca/~lokbani/cours/ift1176/communs/Cours/PDF/beansp2.pdf` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Mica] Microsoft. .net. `http://www.microsoft.com/net` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Micb] Sun Microsystems. Java 2 enterprise edition. `http://java.sun.com/j2ee/` (Dernière mise à jour : 16 mai 2003) (Dernière visite : 24 mai 2003).

[Micc] Sun Microsystems. Java bean. `http://java.sun.com/j2se/1.4.1/docs/guide/beans/index.html` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[MNTN00] Medvidovic, Nenad, Taylor, and Richard N. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1): 70–93, Janvier 2000.

[Ora01] Andy Oram. *Peer to Peer*. O'Reilly, March 2001. ISBN 0-596-00110-X.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[RD01b] Antony I. T. Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.

[Shi00] Clay Shirky. What is p2p...and what isn't. November 2000. `http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html` (Dernière mise à jour : 21 novembre 2000) (Dernière visite : 24 mai 2003).

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. pages 149–160, 2001.

[SP96] Clemens Szyperski and Cuno Pfister. Component oriented programming. In *Workshop on component-oriented programming at ECOOP 1996*, June 1996.

[SSRB01] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects*. Wiley, England, 2001. ISBN 0-471-60695-2.

[Tan02] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, fourth edition edition, 2002. ISBN 0130661023.

[Tis01] James Tisdall. *Beginning Perl for Bioinformatics*. O'Reilly, October 2001. ISBN 0-596-00080-4.

[tpWG] The Peer to-peer Working Group. What is peer-to-peer. `http://www.peer-to-peerwg.org/whatis/index.html` (Dernière mise à jour : indéterminée) (Dernière visite : 15 avril 2003).

[UDD] UDDI.ORG. Universal description, discovery and integration (uddi). `http://www.uddi.org` (Dernière mise à jour : indéterminée) (Dernière visite : 24 mai 2003).

[Wil02] Brendon J. Wilson. *JXTA*. New Riders Publishing, June 2002. ISBN 0-73571-234-4.

[Xul] XulPlanet.com. Xulplanet. `http://www.xulplanet.com` (Dernière mise à jour : 27 avril 2003) (Dernière visite : 24 mai 2003).

Annexe A

Contenu du CD-ROM

Le CD-ROM joint à notre mémoire contient les répertoires suivants:

/memoire/ contenant ce mémoire compilé aux formats Adobe Postscript (PS) et Adobe Portable Document Format (PDF).

/memoire/src/ contenant les sources LaTeX 2_ε et autres fichiers de ce mémoire.

/together/ contenant tous les projets Together Control Center 6.0 réalisés pendant notre stage et la rédaction de ce mémoire.

/dbmain/ contenant tous les projets DBMain réalisés pendant notre stage et la rédaction de ce mémoire.

/demo_prototype/ contenant 3 videos de démonstration du prototype dans 3 cas de figure (1 médiateur et un service, 4 médiateurs en chaîne et un service et pour terminer, 4 médiateurs en boucle et un service).

/implementation/bin/ contenant les fichiers compilés (.class) de notre prototype.

/implementation/src/ contenant les fichiers sources (java, idl et sql) du prototype.

/references/ contenant les spécifications de la norme BSA et le mémoire Laurent Debaisieux and Fernando Desouza [DD03].