Aurélien Francheteau
Brice Francois
Sébastien Helbert
Jérôme Limousin
Jean-Philippe Wilsch

Projet de DESS Génie Informatique

CorbaTrace

Outils d'observation pour applications réparties utilisant CORBA

Faculté des Sciences et Techniques
Université de Nantes
Année 2002/2003

Responsables : Philippe Lamarre
Christian Attiogbé

1 Remerciements

Nous remercions messieurs Philippe Lamarre et Christian Attiogbé pour leurs conseils et leur encadrement durant notre travail.

Nous tenons également à remercier Etienne Juliot, responsable de CorbaTrace lors de l'année précédente, dont l'aide du début jusqu'à la fin du projet nous a été très précieuse.

1.0.0.1

Sommaire

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24 Introduction

Dans ce rapport, nous présentons le travail réalisé cette année sur CorbaTrace, logiciel comprenant un ensemble d'outils d'observation pour le débogage d'applications réparties utilisant CORBA.

Ce logiciel a, dans sa première version, été développé par des étudiants du DESS au cours de l'année 2001/2002. Son but est d'intercepter les échanges de messages sur le bus CORBA, de les enregistrer sous forme de fichiers de logs, puis de traiter ces logs, pour finalement pouvoir en afficher la trace sous forme de diagrammes de séquence. Toutefois cette version présentait des limitations. Tout d'abord les appels de méthodes entre objets locaux n'étaient pas interceptés. Or un appel étant en fait un envoi de message entre deux objets, ils pourrait être utile de les logger pour visualiser les echanges d'informations entre objets. De plus la synchronisation des dates dans les messages interceptés était peu efficace dès lors que les horloges des systèmes répartis étaient désynchronisées. L'ancienne version nécessitait l'utilisation d'un atelier de génie logiciel pour visualiser les diagrammes de séquence générés au format XMI[1] rendant ainsi l'application dépendante d'autres outils, chose parfois très contraignante. Enfin l'aspect austère de l'utilisation du logiciel via la ligne de commande rend l'application difficile d'accès à un utilisateur débutant.

A partir de ces constatations, il nous est demandé cette année d'améliorer le logiciel en travaillant sur plusieurs points. L'utilisation de l'API[2] Java Logging permettra l'interception des messages entre objets locaux et également une interception asynchrone. Il sera nécessaire d'évaluer la qualité actuelle de la synchronisation afin de faire une refonte de l'algorithme en conséquence. Nous devrons réaliser un mécanisme de génération de diagrammes de séquence dans un format nous permettant de les visualiser facilement et fournir un outil de visualisation inclus dans CorbaTrace. Pour terminer nous allons réaliser une interface graphique permettant d'accéder facilement et intuitivement aux principales fonctionnalités de CorbaTrace.

L'objectif est d'obtenir une version complète et stable, d'offrir une distribution accessible à tout programmeur CORBA, et fournir quelques exemples d'applications utilisant CorbaTrace. Dans ce rapport, nous allons tout d'abord détailler le cahier des charges qui nous a été confié. Nous aborderons ensuite Java Logging et la synchronisation des messages. Puis nous présenterons l'interface graphique du logiciel, ainsi que la conception de l'API créée spécialement pour tracer des diagrammes de séquence en fonction des fichiers de logs. Nous terminerons enfin par des tests et l'intégration de CorbaTrace dans un environnement Linux ou Windows.

[1] XMI : XML Metadata Interchange : Définition standardisée de la traduction de modèle UML en XML

[2] API : Application Programming Interface : ensemble de fonctions bas niveau permettant d'élaborer des application de plus haut niveau.

25 Historique

25.1CORBA

Les applications distribuées sont très difficiles à déboguer : le programme utilisant plusieurs machines, maîtriser les échanges d'informations et les accès distants est très complexe. C'est le cas en particulier des applications CORBA.

CORBA est l'acronyme de **C**ommon **O**bject **R**equest **B**roker **A**rchitecture, autrement dit, Architecture Standardisée d'un Négociateur de Requêtes sur des Objets. C'est la solution apportée par l'**OMG** (**O**bject **M**anagement **G**roup) au besoin d'interopérabilité face à la prolifération des machines et des logiciels disponibles sur le marché.

Ses principales fonctionnalités sont :

- La transparence : par rapport au système d'exploitation, au langage de programmation, à la localisation des objets
- CORBA est orienté objet : les principes fondamentaux de la programmation objet sont présent : encapsulation, polymorphisme, héritage et instanciation
- CORBA est orienté services : de nombreux services sont proposés par la norme CORBA pour faciliter son utilisation : nommage, vendeur, événement, notification, …

Nous ne nous étendrons pas plus sur CORBA, de nombreuses présentations existent sur Internet et ce n'est pas le but de ce rapport.

25.2CorbaTrace v0.1

CorbaTrace fut à l'origine un projet de maîtrise intitulé "Outils d'observation pour une application répartie : société BONOM", réalisé par Vincent Tricoire et Frédéric Breton. Il consistait en la réalisation d'outils d'interceptions de messages circulant entre divers objets d'une application répartie utilisant CORBA, et d'outils de visualisation de ces échanges.

L'interception des messages utilisait les intercepteurs portables spécifiés dans la norme CORBA 2.3. Ces intercepteurs, créés sur le bus CORBA, sont indépendants du langage utilisé pour la création des objets et permettent l'interception de l'émission et réception des messages ou d'exception. Les messages interceptés sont enregistrés dans des fichiers journaux. Un fichier journal est crée par objet. Les informations récupérées par les intercepteurs sont l'émetteur et le destinataire du message, un identifiant de message, la date précise d'interception et le contenu du message.

La réalisation des diagrammes de séquence était faite grâce à un paquetage LaTeX. Son utilisation était peu aisée, notamment à cause du fait que les messages n'étaient représentés par une date mais par des décalages de temps.

Plusieurs difficultés s'étaient posées aux deux étudiants durant leur travail, dont :

- ✓ Des problèmes lors de l'identification des clients et des messages (la méthode employée de mettre l'identifiant des objets en paramètre des méthodes était très contraignante)
- ✓ Des problèmes de date dans les messages, les différents objets pouvant avoir des horloges décalées, contrairement à l'hypothèse choisie par les étudiants de même heure pour tous. Il faut pouvoir synchroniser les horloges.
- ✓ Enfin la forme textuelle brute des fichiers de logs et l'utilisation de LaTeX pour le diagramme de séquence ne sont pas satisfaisant. Il serait préférable d'utiliser XML pour l'enregistrement des messages, qui permettrait plus facilement l'application de filtre et la génération de fichier XMI, format permettant d'obtenir une représentation graphique d'un diagramme de séquence dans un atelier de génie logiciel.

25.3CorbaTrace v1.0

Le projet fut repris l'année suivante par cinq étudiants de DESS Génie Informatique. L'objectif était de réaliser une interception moins contraignante et une visualisation plus performante.

L'interception utilise toujours les intercepteurs portables. Des points d'interceptions ont été définis du coté client et du coté serveur :

- ✓ Coté client
  - `send_request` (envoi de requête vers un serveur)
  - `send_poll` (envoi d'une demande d'information au serveur)
  - `receive_repply` (réception d'une réponse)
  - `receive_exception` (réception d'une exception)
  - `receive_other` (réception d'un message qui n'est ni une réponse ni une exception)

- ✓ Coté serveur
  - `receive_request` (réception d'une requête)
  - `receive_request_service_context` (lors de la réception d'une requête, ce point permet de récupérer le ServiceContext, qui permet de passer des informations de l'application hôte aux intercepteurs, de l'intercepteur du client à l'intercepteur du serveur et vice-versa)
  - `send_reply` (envoi d'une réponse à une requête)
  - `send_exception` (envoi d'une exception)
  - `send_other (`envoi d'autre chose qu'une réponse ou une exception)

Les fichiers journaux (nous parlerons de Logs) sont désormais enregistrés sous forme de fichiers XML, format aujourd'hui reconnu dans le domaine de la gestion de données. Ce format, en plus de permettre une lecture humaine aisée, facilite le traitement des fichiers par l'application, grâce à des API prévues pour cela telles que SAX.

Une DTD a été définie pour ces Logs, afin de garantir leur validité. Cette DTD décrit également la forme des filtres, eux aussi au format XML, permettant de sélectionner les informations à afficher dans le diagramme de séquence final.

La partie du programme réalisant cette transformation de fichiers de Logs en diagramme de séquence se nomme Log2XMI. Elle intègre notamment un mécanisme de synchronisation des messages répartis sur des systèmes qui peuvent êtres différents. Celui-ci fonctionne en se basant sur un objet de référence puis en recalculant les dates locales des autres objets par rapport à la date locale de l'objet de référence. Log2XMI intègre également comme dit auparavant un mécanisme de filtrage.

Un diagramme de séquence est donc généré au format XMI, pouvant être lu par des outils de génie logicielle.

L'évolution principale du projet fut sa diffusion sur Internet. Conscient de l'intérêt que ce travail pourrait attirer, l'équipe a publié CorbaTrace sous licence LGPL[3] sur le site http://corbatrace.tuxfamily.org. De nombreuses réactions positives ont été reçues et le site est fréquenté par des utilisateurs de pays tels que l'Allemagne, le Japon ou la Russie.

[3] LGPL : Lesser General Public Licence (Termes et condition à l'adresse suivante : http://www.gnu.org/copyleft/lesser.html)

26 Cahier des charges

26.1 L'existant

Dans cette partie du rapport, nous énonçons le travail déjà réalisé une année auparavant et cela pour mieux comprendre le travail effectué cette année. Pour ceux qui souhaiteraient approfondir cette prise de connaissance, il est conseillé de lire le rapport de l'année passée.

26.1.1Logs distants : CorbaTrace

Les logs distants sur des objets Corba, se base sur des standards de l'OMG sur les intercepteurs. En effet depuis la norme Corba 2.3, une méthode d'interception sur les ORBs a été standardisée. L'interception peut se dérouler sur plusieurs actions : l'émission d'un message, la réception d'un message ou lorsqu'une exception est levée. Les intercepteurs se placent sur le bus Corba et donc sont totalement transparents pour l'application. L'application peut elle par contre agir sur ces interceptions (les autoriser ou non). La figure1 montre l'architecture globale de l'interception :

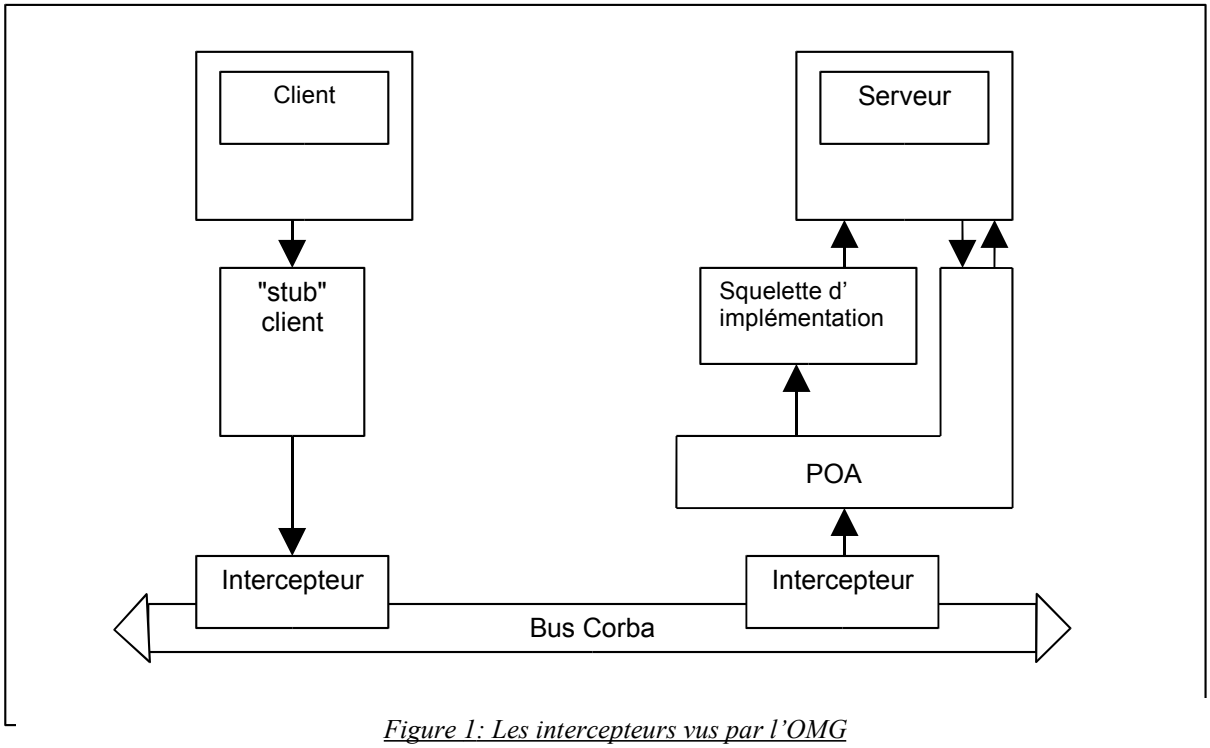


Figure 1: Les intercepteurs vus par l'OMG

Au sein du POA et de l'ORB, l'interception n'est qu'une politique particulière. Un degré d'interception est donc réglable au niveau de chacun. Le degré 0 n'intercepte rien, par contre le degré 1 est suffisant pour intercepter des informations utiles au déboguage.

En ce qui concerne Corbatrace, des classes spécifiques à l'interception ont été surchargées ou encapsulées de manière à minimiser le nombre de changements nécessaires à apporter à une application pour logger les communications entre les différentes parties. Les intercepteurs s'enregistrent auprès de l'ORB via ces classes. L'activation de l'intercepteur est par contre différente suivant qu'il est fait du coté client ou du côté serveur. Une fois l'intercepteur enregistré, il faut le placer sur le bon composant. Pour le cas du client il est mis sur l'ORB, par contre pour le cas du serveur il est mis sur le POA. Une fois que l'intercepteur est placé sur le POA, un autre POA est créé dynamiquement avec les intercepteurs activés à partir d'un POA existant dans l'application cliente (rootPOA). La figure 2 représente des intercepteurs vus de l'application Corbatrace.

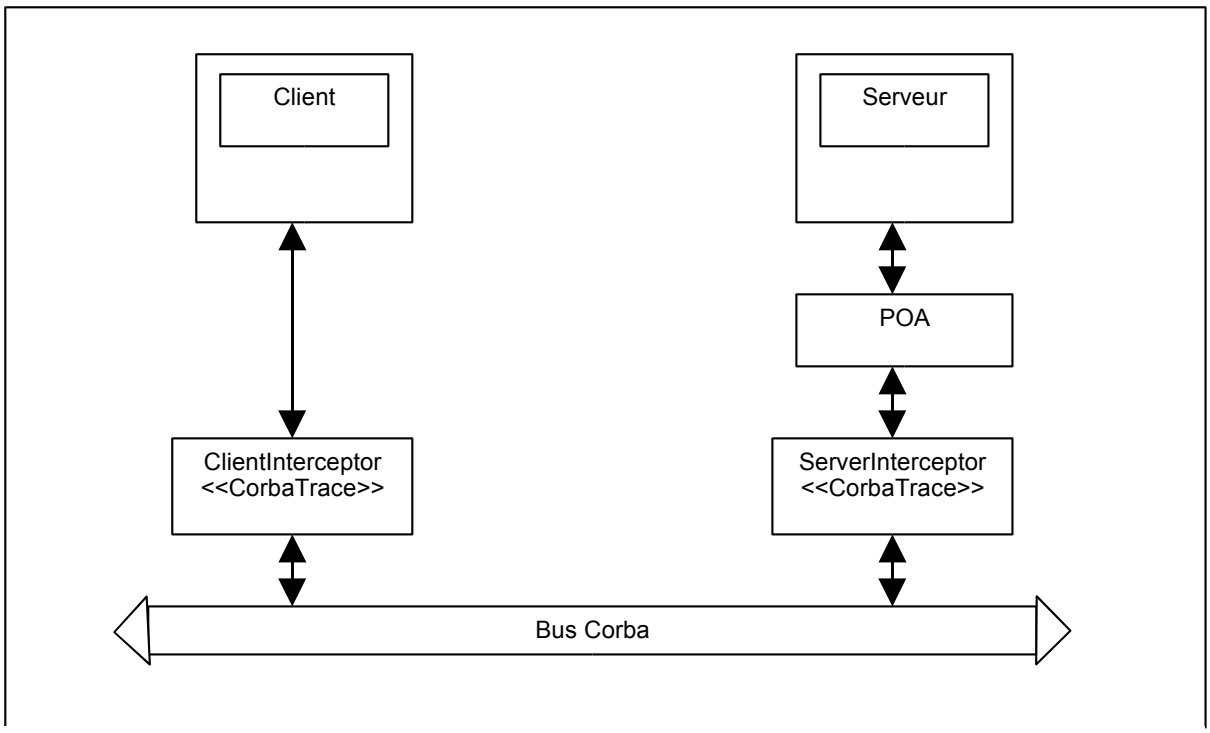


*Figure SEQ Figure * ARABIC 2: Les intercepteurs vus par CorbaTrace*

Après toutes ces initialisations, dans le cas client, on peut utiliser l'objet métier directement. Par contre pour le cas serveur, en plus d'utiliser l'objet métier, il faut positionner l'objet métier servant sur le nouveau POA créé dynamiquement par la classe IntercepteurServeur. Pour plus d'information sur ce sujet vous pouvez vous reporter au rapport du projet Corbatrace V1.0.

Comme nous l'avons énoncé précédemment, plusieurs actions sont possibles à intercepter. L'envoi d'une requête implique grâce au mécanisme d'interception initialisé, la mise en route de celui-ci. L'interception se fera du coté client (send_request). La requête sera ensuite interceptée du coté serveur en deux étapes, la première (receive_request_service_context) qui permet de récupérer les informations de l'émetteur de la requête. Ensuite il y a l'interception effective de la réception de la requête (receive_request). Une fois la requête exécutée au niveau du serveur, il y a une interception de l'envoie de la réponse (receive_request) et enfin une interception au niveau du client de la réception de la réponse (receive_reply). Si l'exécution de la requête lève une exception par le serveur, l'envoie de cette exception au client sera interceptée au niveau serveur (send_exception) puis à sa réception au niveau client (receive_exception). Le client peut aussi envoyer un type de requête particulier. Il consiste en une demande d'informations du client sur l'état du servant ou sur l'ORB. Ce type de demande sera intercepté au niveau client (send_poll), par contre pour la suite cette demande sera traitée comme toutes les autres requêtes. Il y a enfin un dernier type d'interception, l'envoie côté serveur (send_other) et la réception côté client (receive_other) de tout autre chose qu'une réponse de requête ou une exception.

Voici un schéma récapitulatif des différents types d'interception :

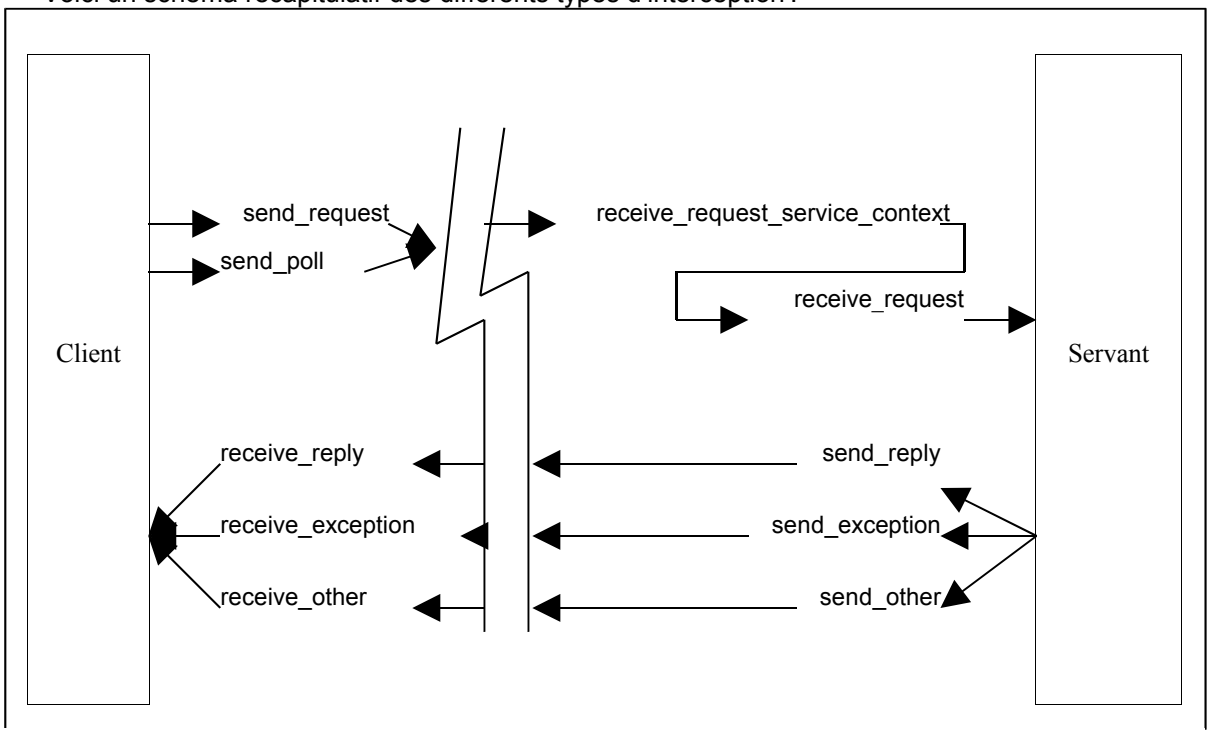


*Figure SEQ Figure * ARABIC 3: Les points d'interception*

Une fois les interceptions générées, il faut traiter les informations qu'elles produisent.

26.1.2Log2XMI

Pour exploiter les fichiers de logs issus des interceptions de CorbaTrace, l'application Log2XMI a été réalisée. Elle réalise la transformation des fichiers de logs, au format XML, en un seul fichier au format XMI, une DTD standardisée par l'OMG, pouvant être utilisée pour représenter des diagrammes de séquences, forme sous laquelle vont être représentés les échanges de messages dans l'application CORBA loguée.

5 étapes distinctes composent le processus de transformation de Log2XMI :

- Le parsage des fichiers de logs, contenants uniquement des demi-messages, c'est à dire les informations sur uniquement l'envoi ou la réception d'un message.
- La fusion des demi-messages, pour obtenir un message complet à partir de deux demi-messages correspondant.
- La synchronisation des messages, pour corriger les différences issues de décalage entre les horloges des différents hôtes des objets de l'application CORBA loguée.
- Le filtrage des messages, pour faire apparaître sur le diagramme de séquence uniquement les informations que l'utilisateur juge pertinentes.
- La création du fichier XMI, à partir des messages filtrés

26.1.3La synchronisation

Dans sa première version, Corbatrace était déjà doté d'une synchronisation des messages logués, partie essentielle du logiciel sans laquelle toute visualisation d'une trace aurait été très peu compréhensible et donc sans intérêt.

La principale difficulté vient des différences entre les horloges d'objets distants.

Les programmeurs avaient alors posé leur problématique et envisagé une technique de synchronisation y répondant, avant d'implémenter leur solution.

Dans la problématique, l'hypothèse est faite que les messages sont instantanés et le traitement des messages est proposé.

La technique de synchronisation envisagée comporte trois étapes successives:
- l'ajout des messages logués
- l'estimation des décalages d'horloges
- la génération des messages séquentialisés

L'implémentation a aboutit bien qu'elle eut pu être améliorée (notamment dans la deuxième des étapes énoncées ci-dessus).

26.2 Apports nécessaires

26.2.1Création du fichier de journal (Logs)

La version 1.0 de CorbaTrace utilisait la classe `IndentString` du package `corbatrace.utils` pour enregistrer des Logs au format XML dans un « buffer » pour ensuite écrire son contenu dans un fichier de Log.

Cette manière de procéder a de nombreux désavantages. D'abord, les performances de l'intercepteur sont réduites. En effet, à chaque interception sur le bus CORBA, les informations obtenues sont stockées dans des objets de type XMLLog qui sont transformés au format XML au moment de l'écriture dans le fichier de Log. Or cette transformation est coûteuse en temps, d'autant plus que XML à pour caractéristique d'être « verbeux ». De même l'écriture dans le fichier de log nécessite une ouverture de fichier, un positionnement en fin de fichier, l'écriture du flux XML dans ce fichier, puis sa fermeture. Autant d'opérations qui sont bien connues pour êtres coûteuses en temps. Ainsi durant toutes ces opérations l'application contenant les intercepteurs est bloquée. Même si ce temps reste relativement faible, il ne faut pas oublier que CorbaTrace est un notamment un outil de déboguage et se doit de fournir les informations les plus précises que possibles en ce qui concerne la succession des évènements dans le temps.

De plus si l'intercepteur s'interrompt de manière brutale lors de l'écriture du « buffer », les fichiers générés seront invalides. Comme nous venons de la préciser CorbaTrace est un outil de déboguage, les applications qui l'utilisent sont donc potentiellement plus sujettes à la possibilité de terminaison inattendue du programme que d'autres. Si cela se produit alors que l'intercepteur écrit dans le fichier de Logs, le fichier risquera d'être invalide (non respect de la DTD car fichier non complet).

Pour ces deux raisons nous avons besoin d'un mécanisme asynchrone qui se charge de formater les Logs et de les écrire dans un fichier indépendamment de l'application contenant les intercepteurs. Nous allons donc déléguer cette tâche à un package spécialement conçu pour cela, intégré depuis la version 1.4 de Java, le package JavaLogging.

26.2.2Synchronisation

Cette présentation des apports envisagés pour la synchronisation est un préambule au chapitre qui lui est consacré (chap. 5). Les modifications réellement effectuées (ou à effectuer) découlant de notre analyse, sont énoncées et détaillées dans ce chapitre.

Etant donné que la nouvelle version de Corbatrace ne se contente plus de gérer le seul cas des logs Corba, mais s'intéresse désormais également aux logs locaux, il s'agit d'étudier ce nouveau cas: nécessite t'il des modifications dans la synchronisation? Nous allons donc effectuer une phase d'analyse, suivie des éventuels apports nécessaires.

Il s'avère aussi que l'implémentation de la technique de synchronisation, bien qu'étant réalisée, comportait de légères lacunes particulièrement lorsque les horloges des différents systèmes répartis n'étaient pas en concordance, qu'il s'agit maintenant de combler. Un affinage de l'estimation des décalages d'horloge était d'ailleurs proposé dans le rapport de l'année précédente.

Un autre travail envisagé serait de reconsidérer les hypothèses de départ. Ce serait d'abord un sérieux travail d'analyse, qui signifierait surtout en cas de nouvelles hypothèses de travail de reprendre une grande partie du code réalisé pour le mettre à jour. Nous serions tout de même fortement intéressés par la gestion du parallélisme qui semble malheureusement avoir été laissée de côté.

Dans tous les cas, il est nécessaire de rependre le code afin de l'améliorer. Les méthodes de *refactoring* permettraient de le rendre plus clair, plus accessible et ainsi plus facile à modifier.

26.2.3 Interface graphique

Nous avons également constaté qu'un frein au développement de CorbaTrace auprès d'un plus grand public pouvait être une certaine difficulté d'approche et de démarrage du logiciel, comme par exemple le fait que l'application ne fonctionne uniquement via la ligne de commandes.

Afin de mettre CorbaTrace à la portée de plus de personnes, une solution pouvait être de réaliser une interface graphique accompagnant le logiciel. Cette interface proposerait une aide notamment pour l'utilisation de l'application Log2XMI, en proposant par exemple :

- De pouvoir récupérer des fichiers de logs situés sur des machines distantes, cas pouvant être fréquent puisque CorbaTrace propose d'aider au débogage d'applications CORBA, donc pouvant être réparties.
- D'aider à l'écriture de fichiers de filtres. Ces fichiers devaient auparavant être écrits à la main. L'interface proposerait à l'utilisateur de générer le fichier XML de filtre à partir d'informations sélectionnées dans des menus déroulant, très simples d'utilisation.
- D'utiliser Log2XMI plus facilement, simplement en sélectionnant les fichiers de logs, le fichier de filtre et les options que l'utilisateur désirent pour générer un diagramme de séquence avec des listes, des cases à cocher, des widgets graphiques très simples à utiliser.
- Permettre à l'utilisateur de visualiser ses diagrammes de séquence sans recours à un logiciel externe en proposant un visualiseur de fichier SVG

Pour garder l'aspect important de portabilité de CorbaTrace grâce à l'utilisation du langage Java, l'interface graphique sera développée grâce à la librairie Swing, l'API graphique de Java [12].

26.2.4 Diagrammes de séquence

La précédente version du projet génère des diagrammes de séquences sous la forme de fichiers XMI. Cela permettait un affichage des diagrammes dans les logiciels de modélisation UML tel que *Poseidon for UML* [1], *Rational Rose* [2] ou *MagicDraw UML* [3]. L'ancienne application peut également convertir les fichiers XMI en fichier TeX affiché sous *LaTeX* [4].

L'inconvénient majeur de ces choix est qu'il faut soit disposer d'un logiciel de modélisation UML alors que l'on veut simplement afficher un diagramme et non le modéliser ou soit visualiser le diagramme sous LaTex qui donne un résultat médiocre la librairie fournie étant trop limitée pour répondre à nos besoins. En plus d'être inadaptées aux besoins, ces solutions sont peu portables (tout le monde ne dispose pas d'outils de modélisation UML ou de LaTeX).

Il nous a donc semblé important d'utiliser un autre outil de visualisation plus adapté. Nous nous sommes tournés vers SVG [5] et en expliquons les raisons dans la partie 6 du rapport.

27 Java Logging

27.1Pourquoi utiliser Java Logging ?

27.1.1Principe de Java Logging

Les applications font des appels de méthodes sur des Objets de la classe `Logger`. Ces objets créent des enregistrements de Logs, les `LogRecord` qui sont envoyés aux objets de la classe `Handler` pour être publiés. Les objets de la classe `Logger` et `Handler` peuvent utiliser des niveaux pour filtrer les Logs suivant leur importance. Ce sont les objets de type `Filter` qui se chargent de filtrer les Logs. Enfin, les objets de type `Handler` peuvent utiliser des instances de `Formatter` pour mettre en forme le flux de sortie.

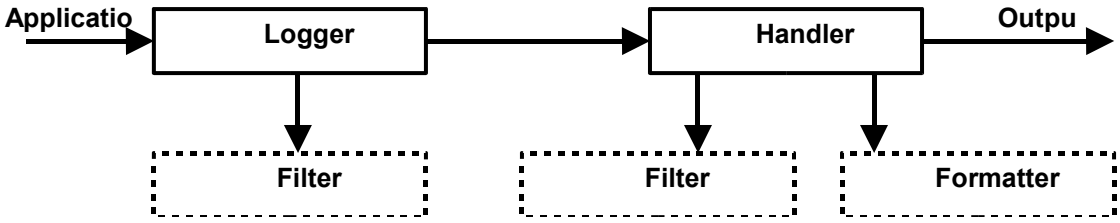


Figure 2:Principe de Java Logging

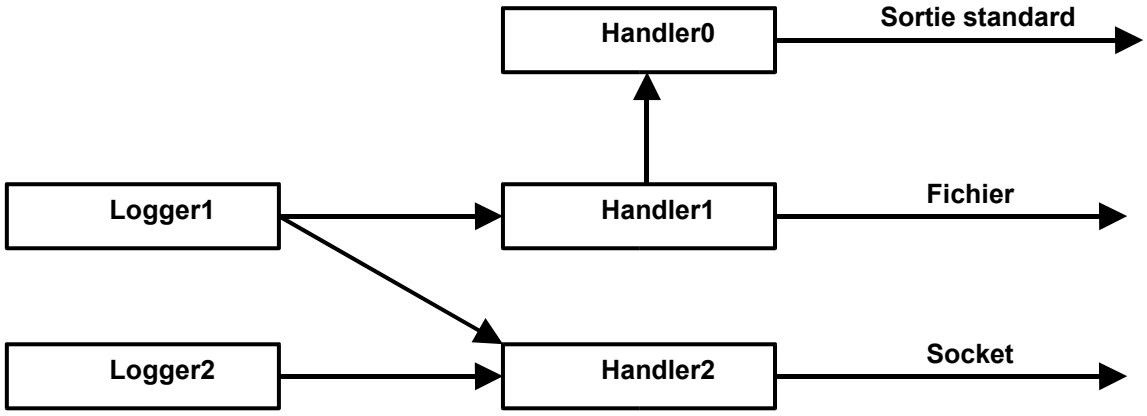


Figure 3:Principe de Java Logging

Chaque Logger peut posséder plusieurs `Handler`. Les `Handler` peuvent êtres organisés de façon hiérarchique et répercuter la publication des Logs aux `Handler` de niveaux supérieurs.

Seul le traitement du Logger est bloquant pour l'application. Le Logger doit être conçu de façon à minimiser le coût et le temps avant de transmettre aux `Handler` les enregistrements de Logs, qui eux, appartiennent à un autre processus que celui de l'application. Les opérations les plus coûteuses telles que la localisation de la sortie pour le flux et son formatage sont donc prises en charge par les `Handler` de façon asynchrone afin d'optimiser les performances de l'application.

27.1.2Utilisation de Java Logging

Nous devrons étendre la classe Logger pour permettre par le simple appel de méthodes statiques sur cette classe de loguer des informations. Nous n'aurons pas recours aux filtres.

Nous ne pouvons pas utiliser le formateur XML proposé par défaut par Java 1.4 pour deux raisons. La première est que ce formateur logue des enregistrements de type LogRecord qui contiennent des informations telles que la date, un numéro de séquence un identifiant de Thread émetteur, etc. qui sont soit redondantes par rapport aux informations que nous disposons déjà, soit superflues (en tous cas pour le moment). Le seconde raison est qu'il autorise de loguer une chaîne de caractères dans un champ message,

mais les caractères spéciaux XML que peut contenir cette chaîne seront déspécialisés avant l'émission du flux de sortie.

Ce qui implique lors de la lecture du fichier de Logs de parser chaque Log Java pour récupérer les informations Corba qu'ils contiennent avant de les convertir au format XML pour de nouveau les parser, donc deux DTD seront nécessaires, etc. Cette solution n'est pas envisageable. Nous choisissons de surcharger la Classe XMLFormatter pour créer notre propre Formateur, instance de la classe CXMLFormatter, qui se chargera de formater des enregistrements regroupant les informations minimales nécessaires aux Logs CorbaTrace des instances de CLogRecord.

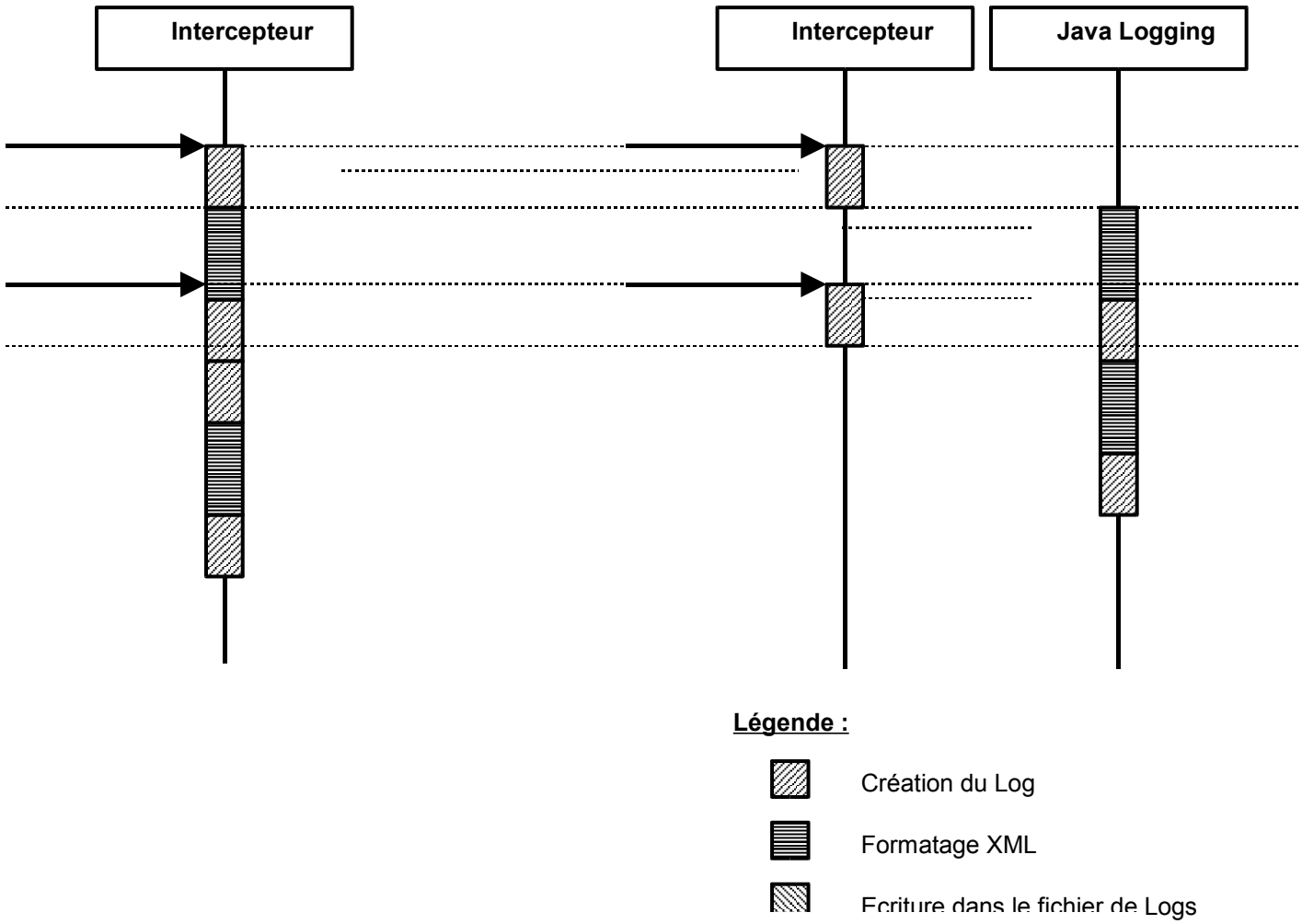


Figure 4 : Interception

27.1.2.1 CorbaTrace Logger : Clogger

Cette classe surcharge La classe Logger de JavaLogging et fourni un ensemble de méthodes permettant d'enregistrer les Logs distants propres à CORBA mais aussi des Logs locaux. De plus c'est auprès de cette classe que se feront les associations identifiant Objet/Nom d'objet dont nous reparlerons plus tard (cf. "27.2.2 La table de correspondance"). Les signatures des méthodes sont les suivantes :

```
public static void clearNames();
public static String setName(Object object, String name);
public static String getName(Object object);
public static void logCallBegin(String loggerName, Object src, Object dest,
                                                String method, String[] args);
public static void logCallEnd(String loggerName, Object src, Object dest,
                                              String method, String result);
public static void logActivityBegin(String loggerName, Object obj);
public static void logActivityEnd(String loggerName, Object obj);
public static void logTrace(String loggerName, Object obj, String message);
public void log(LogRecord record);
```

Les trois premières méthodes permettent de manipuler la table de correspondance dont nous avons parlé précédemment, les cinq suivantes concernent les Logs locaux et permettent, respectivement de déclarer l'appel d'une fonction, la fin de son exécution, le début d'activité d'un thread, la fin de son activité et enfin de construire un log de trace. La dernière méthode est utilisée par les intercepteurs CORBA qui construisent eux même leur propre Log, le CLogRecord.

Remarque : Tous les Logs CorbaTrace sont générés via la méthode "log" de cette classe. En effet, les méthodes de création de Logs locaux créent en fait une nouvelle instance de la classe CLogRecord, modifient les attributs appropriés de l'enregistrement puis invoquent la méthode "log" en le passant en paramètre. Ceci permet notamment d'attribuer à chaque log un identifiant unique.

27.1.2.2 CorbaTrace LogRecord : CLogRecord

La classe CLogRecord étend la classe LogRecord du package JavaLogging. Elle permet de construire des objets dans lesquels peuvent êtres stockés toutes les informations enregistrées par les intercepteurs CORBA. Cette classe est donc très similaire à la classe XMLLog de CorbaTrace v1.0.

27.1.2.3 CorbaTrace XMLFormatter : CXMLformatter

La classe CXMLFormater étend la classe XMLFormater du package JavaLogging. Elle permet de transformer un objet de type CLogRecord au format XML correspondant à la DTD

27.2Choix et modifications effectuées

27.2.1Classe IndentString :

La classe `IndentString` qui se situe dans le package `corbatrace.utils` à été étendue pour simplifier son utilisation. Elle à pour but de créer une chaîne de caractères en permettant une indentation correcte des balises XML de manière simple et performante. L'indentation ayant pour seul intérêt de rendre plus aisée la lecture des informations des fichiers de Logs qui n'apparaîtraient pas dans les diagrammes de séquences générés. Dans la version 1.0 de CorbaTrace cette classe proposait les quatre méthodes suivantes :

- `inc` : Incrémente le niveau d'indentation
- `dec` : Décrémente le niveau d'indentation
- `attribute` : Concatène le texte
- `newLine` : insère un saut de ligne
- `insert` : indente, insère le texte et passe à la ligne

Nous avons choisi d'étendre cette classe pour simplifier le code nécessaire et donc limiter le nombre d'erreurs potentielles et rendre transparente la gestion des indentations et des sauts de lignes.

Les méthodes ajoutées sont les suivantes :

- `openTag` : Ouvre une balise XML
- `closeTag` : Ferme une balise XML
- `attribute` : ajoute un attribut
- `beginTag` : insère une balise XML ouvrante
- `endTag` : insère une balise XML fermante

Ainsi pour obtenir le fichier XML suivant :

```
<personne>
   <age>53</age>
   <name first="Omer" last="Simpson">
</personne>
```

le code nécessaire sera :

```
out.beginTag("personne");
out.beginTag("age");
out.insert("53");
out.endTag("age");
out.openTag("name");
out.attribute("first", "Omer");
out.attribute("last", "Simpsons");
out.endTag("name");
out.endTag("personne");
```

Les méthodes `inc`, `dec`, `append` et `newLine` deviennent obsolètes.

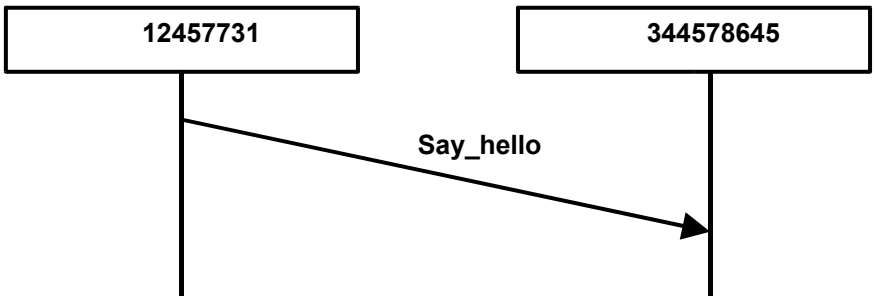
27.2.2 La table de correspondance

Lorsque des Logs sont générés nous leur attribuons un identifiant unique correspondant à l'objet qui à émis le log. Cet identifiant permettra par la suite au générateur de diagrammes de séquences de pouvoir identifier les émetteurs et récepteurs de chaque Log.

Cependant ces identifiants uniques correspondent à des entiers qui n'ont pas de signification particulière pour l'utilisateur (cf. Figure 5). Il nous à donc fallu mettre en place un mécanisme permettant au

programmeur d'associer aux identifiants des objets émetteurs ou récepteurs des noms (chaînes de caractères) que nous appellerons table de correspondance.

Figure 5 : Exemple de diagramme sans table de correspondance



Pour associer un nom à un objet il suffit d'invoquer la méthode statique `setName` de la classe `Clogger`.

Ainsi en insérant le code nécessaire avant de générer des Logs, le diagramme de séquence devient plus explicite (cf. Figure 6).

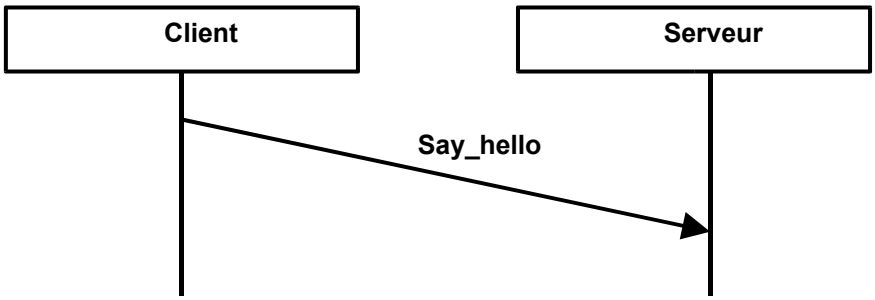


Figure 6 : Exemple de diagramme avec table de correspondance

Il est possible d'initialiser la table de correspondance en utilisant la méthode `Clogger.clearNames` et de retrouver le nom associé à un objet en invoquant la méthode `Clogger.getName`. Evidement, il est à la charge du programmeur de s'assurer que les noms donnés identifient les objets de manière unique, sans quoi le diagramme de séquence généré risque d'être incohérent.

27.2.3Nouvelle DTD pour les Logs

L'ajout de Logs Locaux pour CorbaTrace v2.0 a imposé la modification de la DTD des Logs. Les modifications sont assez peu nombreuses et ne font que rendre certains champs optionnels. Cela permet de garantir une compatibilité ascendante avec les versions précédentes de CorbaTrace.

Les modifications apportées sont les suivantes :

- Le champ opération du champ message est rendu optionnel. Ce champ n'étant pas nécessaire pour les Logs de début et de fin d'activité.

CorbaTrace v1.0

```
<!ELEMENT message(
      local_object,
      distant_object?,
      operation,
      result?,
      options?
   )>
```

→

CorbaTrace v2.0

```
<!ELEMENT message(
      local_object,
      distant_object?,
      operation?,
      result?,
      options?
)>
```

- L'attribut `request_id` du champ Message devient optionnel. Ce champ qui identifie les requêtes sur le bus CORBA de manière unique, n'a pas de signification pour les Logs locaux.

CorbaTrace v1.0

CorbaTrace v2.0

```
<!ATTLIST message
      mesg_id CDATA #REQUIRED
      request_id CDATA # REQUIRED
      type CDATA #REQUIRED
>
```

→

```
<!ATTLIST message
      mesg_id CDATA #REQUIRED
      request_id CDATA #IMPLIED
      type CDATA #REQUIRED
>
```

- Les attributs `date` et `request_id` du champ `distant_object` deviennent optionnels. Le premier parce que pour les logs locaux, on considère que le temps qui s'écoule entre l'appel de méthode et le début de l'exécution de la méthode par l'objet est nul. Le second pour la même raison que celle citée pour l'attribut `request_id` du champ message.

CorbaTrace v1.0

```
<!ATTLIST distant_object
      id CDATA #REQUIRED
      date CDATA #REQUIRED
      request_id CDATA #REQUIRED
>
```

→

CorbaTrace v2.0

```
<!ATTLIST distant_object
      id CDATA #REQUIRED
      date CDATA #IMPLIED
      request_id CDATA #IMPLIED
>
```

- L'attribut `inout` du champ argument devient optionnel puisqu'en Java tous les attributs d'appels sont considérés comme Entrée et Sortie.

CorbaTrace v1.0

```
<!ATTLIST argument
      inout(in|out|inout)#REQUIRED
      name CDATA #IMPLIED
      value CDATA #REQUIRED
      type CDATA #REQUIRED
>
```

→

CorbaTrace v2.0

```
<!ATTLIST argument
      inout(in|out|inout)#IMPLIED
      name CDATA #IMPLIED
      value CDATA #REQUIRED
      type CDATA #REQUIRED
  >
```

27.2.4Appel de procédures

Le premier type de log local inclus dans l'application CorbaTrace est l'appel de procédure qui est constitué en fait de l'appel de la méthode et de la fin de la méthode. Nous avons choisi d'ajouter ces fonctionnalités car elles correspondent a celle proposée par CorbaTrace V1.0 mais pour les objets distants. L'appel de ces deux logs se fait, contrairement aux appels distants, par le développeur. En effet nous n'avons pas fait un intercepteur local au niveau de la machine JAVA pour récupérer tous les appels de méthode. Le programmeur devra utiliser la classe "CLogger" du package "`corbaTrace.logger`". Les deux appels sont :

- `logCallBegin(String loggerName, Object src, Object dest, String method, String[] args)`

`loggerName` : nom donné au `logger`, de type `Sring`.

`src` : objet source, celui qui appel la méthode, de type Objet .

`dest` : objet destination, celui sur qui l'appel est fait, de type Objet.

`method` : nom de la méthode appelée, de type String.

`args` : tableau d'arguments passés à la méthode.

- `logCallEnd(String loggerName, Object src, Object dest,String method, String result)`

`loggerName` : nom donné au logger, de type Sring.

`src` : objet source, celui sur qui l'appel est fait, de type Objet.

`dest` : objet destination, celui qui appel la méthode, de type Objet.

`method` : nom de la méthode appelée, de type String.

`result` : resulat de l'appel de la méthode, de type String.

Pour la visualisation de ce type de message sur le diagramme de séquence nous suivrons la standardisation de l'OMG. Celle-ci consiste-en :

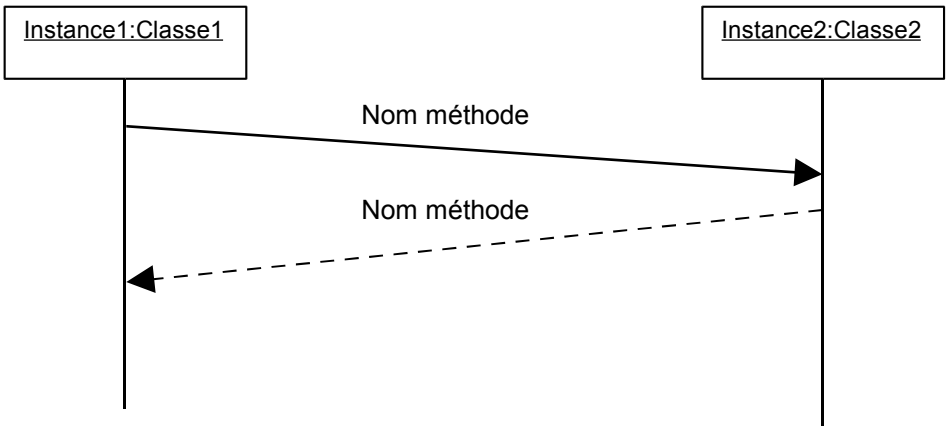


Figure 7: diagramme de séquence avec appel et fin de méthode

Cependant, comme c'est le développeur qui insère ces appels de log dans son programme, on ne pourra l'empêcher de ne loguer qu'une partie de l'appel de procédure; soit l'appel en tant que tel, soit uniquement le retour de la méthode.

Dans une prochaine version nous pourrions ajouter en plus du nom de la méthode le résultat de celle-ci. Nous ne l'avons pas fait pour ne pas surcharger le diagramme.

27.2.5Début et fin d'activité

Un autre type de log local est inclus dans CorbaTrace, il s'agit du début et fin d'activité. Il permet au développeur de signaler lorsqu'un processus débute son activité, et lorsqu'il la termine. Ceci est très utile car dans les applications réparties, il est courant de rencontrer de la programmation multi-processus. Le développeur peut ainsi surveiller l'état de ses processus pour confirmer le bon fonctionnement de son application. Comme pour l'appel de méthode, c'est le développeur qui doit indiquer dans son code ou commence et finit l'activité de son processus. Le programmeur devra utiliser la classe "CLogger" du package "`corbaTrace.logger`". Les deux appels sont :

- `logActivityBegin(String loggerName, Object obj)`

`loggerName` : nom donné au `logger`, de type String.

`obj` : l'objet qui commence son activité, de type Objet.

- `logActivityEnd(String loggerName, Object obj)`

`loggerName` : nom donné au `logger`, de type String.

`obj` : l'objet qui fini son activité, de type Objet.

Pour la visualisation de ce type de message sur le diagramme de séquence nous suivrons la standardisation de l'OMG. Celle-ci consiste-en :

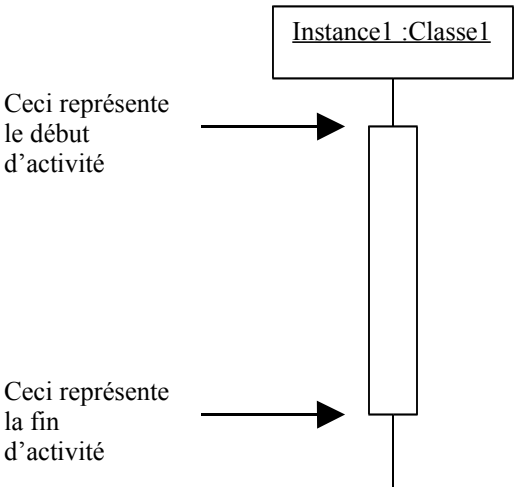


Figure 8: diagramme de séquence avec début et fin d'activité.

Étant donné que le développeur indique ces débuts et fins d'activité, on se rend vite compte que l'on pourra tomber dans des cas incohérents. En effet si le programmeur ne positionne pas bien ces logs ou n'en met pas le bon nombre, on va se trouver devant un diagramme qui ne reflètera pas la réalité. Voici un listing non exhaustif des cas qui posent problème :

Cas 1 : qu'un seul log de début ou de fin de méthode :

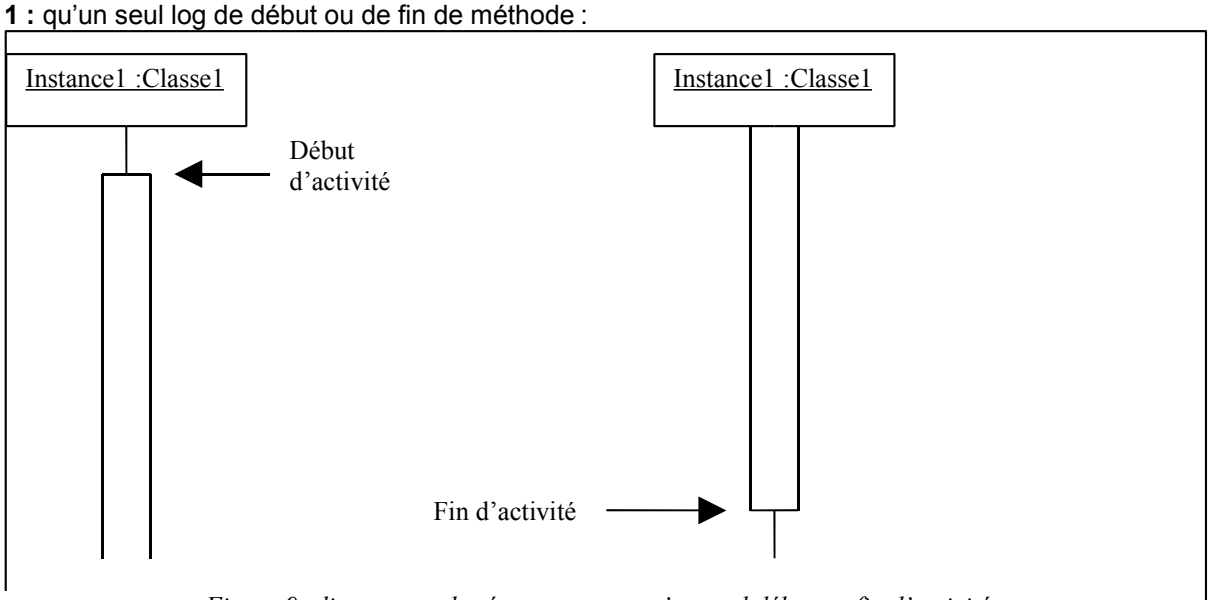


Figure 9: diagramme de séquence avec qu'un seul début ou fin d'activité.

Dans ce cas, il n'y a qu'un seul début ou fin d'activité. On ne peut donc pas savoir quand commence ou fini le processus. La représentation comme elle est sur le diagramme de séquence est une représentation erronée de la réalité. Nous avons déduit qu'il finissait soit après la fin du diagramme ou commençait avant le début du diagramme.

Cas 2 : qu'un log de début ou de fin pour deux de début ou de fin:

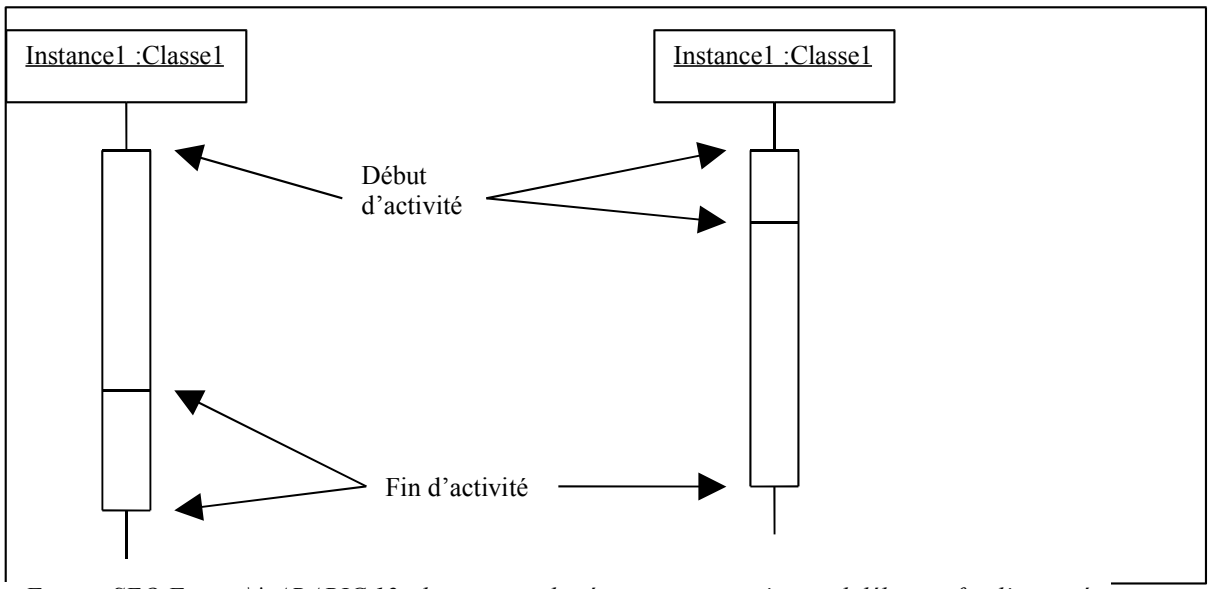


*Figure SEQ Figure * ARABIC 12: diagramme de séquence avec qu'un seul début ou fin d'activité pour deux fins ou débuts d'activité.*

Dans ce cas si le développeur indique un début ou fin d'activité pour deux fins ou débuts d'activités, l'information a affiché sera donc encore erroné. Nous avons relié tous les début et fin d'activité pour respecter les normes UML. Ceci est donc inconcevable.

Cas 3 : deux logs de débuts ou de fins pour deux de début ou de fin:

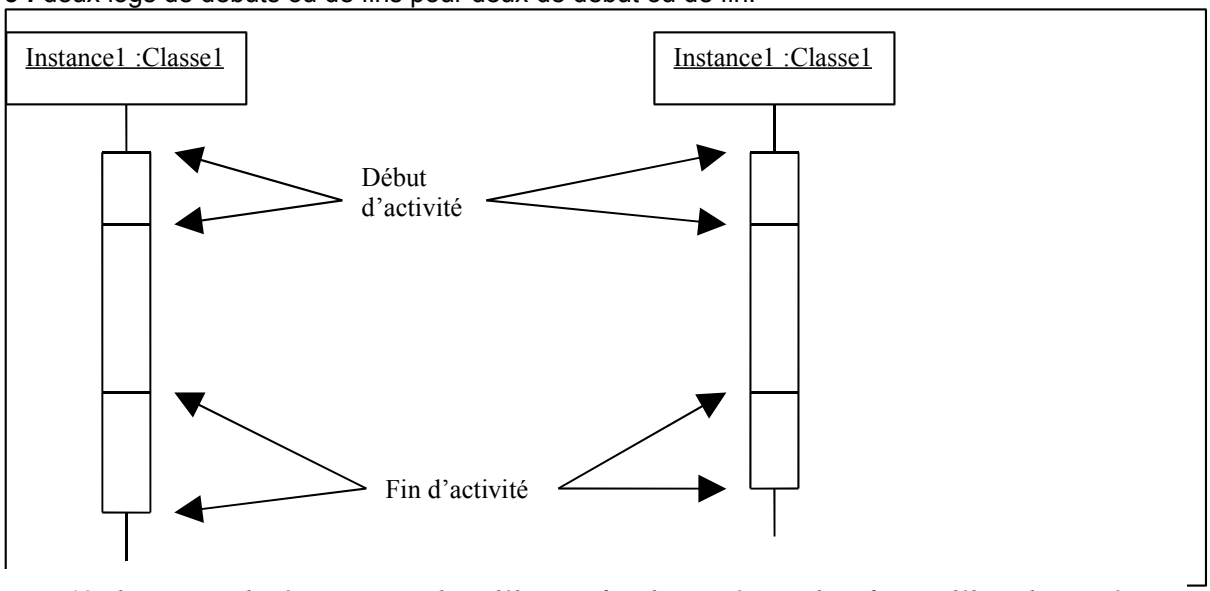


Figure 10: diagramme de séquence avec deux débuts ou fins d'activité pour deux fins ou débuts d'activité

Dans ce cas une activité d'un processus est encapsulée dans une autre activité. Ceci est totalement impossible, nous ne pouvons donc pas nous permettre d'afficher une incohérence. Ces cas ne sont pas exhaustifs il en existe bien d'autres, mais nous n'allons pas tous les monter. Voici par contre un cas cohérent qui pourra être affiché.

Cas 4 : cohérence entre log de début et de fin d'activité:

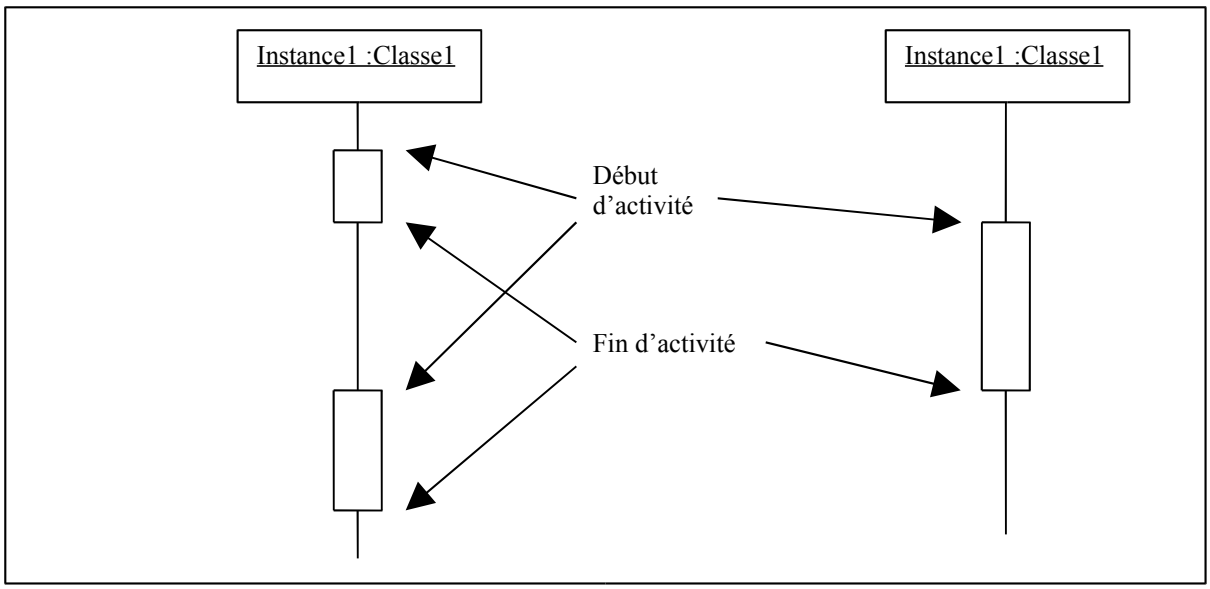


Figure 11: diagramme de séquence cohérent pour les débuts et fins d'activité

Dans ces deux cas Les débuts de fin d'activités sont cohérents. On peut aisément les afficher tout en étant quasiment sur de montrer au développeur ce qu'il attend. Pour tous les autres cas d'incohérence, nous avons choisi de lever une exception qui sera interceptée par l'interface graphique et qui affichera un message au développeur pour lui indiquer l'incohérence. Ensuite sur le diagramme, nous n'afficherons que les états cohérents. Nous aurions très bien pu afficher tous les états, même ceux incohérents, et laisser le développeur se rendre compte de l'incohérence et de modifier ses logs dans son application. Cependant nous pensons que ceci induirait le développeur en erreur. Qui plus est, notre souhait de respecter la norme UML n'aurait pas été exaucé.

27.2.6Messages de trace

Le dernier type de log local que nous avons ajouté à notre application CorbaTrace, est le message de trace. En effet lors du déboguage d'une application il est très souvent nécessaire de pouvoir afficher le contenu d'une variable à un endroit donné du programme. Plutôt que de faire un traditionnel "`System.out`" qui est vivement déconseillé, nous permettons au développeur de directement l'intégrer dans le diagramme de séquence par l'intermédiaire d'un log trace. Le programmeur utilisera la classe "`CLogger`" du package "`corbaTrace.logger`". L'appel est :

```
logTrace(String loggerName, Object obj, String message)
```

`loggerName` : nom donné au logger, de typa String

`obj` : objet dans lequel est appelée la méthode, de type Objet

`message` : message à afficher, de type String

La visualisation de ce type de log se fera par l'intermédiaire des commentaires en UML. Voici un exemple de ce type de log sur un diagramme de séquence :

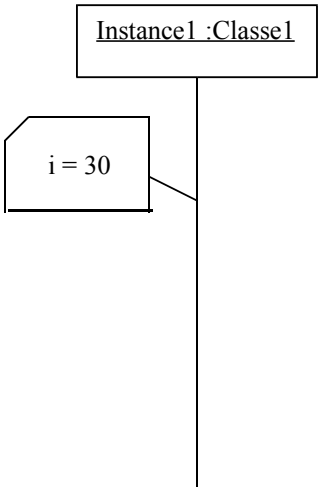


Figure 12: diagramme de séquence avec un message trace

Dans ce dernier type de log il n'y a à priori pas d'incohérence possible. Par contre si le message est trop long à afficher, alors le diagramme peut devenir très vite illisible. Ce type de log doit être utilisé par le développeur uniquement pour des informations essentielles et pertinentes.

27.3Messages Locaux

27.3.1 Description du besoin

L'application CorbaTrace permet d'intercepter les communications Corba et de les visualiser. Ceci est très utile pour le déboguage des applications réparties. En effet, dans une application de ce type, il est très difficile de visualiser les traces qu'apporte un déboguage classique sur tous les objets distants.

L'intérêt des traces dans une application est de voir leur séquencement pour être sur que l'application fait bien ce que l'on veut. CorbaTrace permet de visualiser les appels de méthodes distants mais pas locaux. Ceci est donc une amélioration à ajouter aux fonctionnalités de CorbaTrace. Ainsi pour une application répartie, la visualisation des appels distants et des appels locaux sera possible. Ce qui fournira un outil performant et très utile à tous les développeurs d'application répartie. Pour la visualisation on ne distinguera pas un objet distant d'un objet local. Ce masque facilitera la visualisation des informations de déboguage. Comme nous vous l'avons décrit auparavant, les logs locaux sont de trois formes différentes. Les appels et retour de méthodes, les débuts et fins d'activités et les traces constituent les informations que peuvent décrire les logs locaux. Cet ensemble pourra être enrichi ultérieurement.

28 Synchronisation des messages

Pour commencer, il s'agit de resituer le problème de la synchronisation. Sur un ORB, les objets sont distants, ce qui signifie que rien ne garantit une horloge commune. Pour diverses raisons, les horloges peuvent avoir des réglages différents (fuseaux horaires, mauvais réglage...).

Notre travail est donc d'estimer ce décalage d'horloge entre les différents objets afin de mettre à jour les dates d'envoi et de réception des messages et finalement d'obtenir ainsi une trace cohérente. Dans la version 1.0 de CorbaTrace, une synchronisation a été réalisée *(cf. 2.1.3)*, et c'est celle-ci que nous reprenons et améliorons avant de l'intégrer dans la version 2.0 .

28.1Reprise de l'existant

De l'étude du travail qui a déjà été fait, nous avons constaté plusieurs points intéressants:

- Le code comporte une **structure de graphes** tout à fait réutilisable et extensible à de nouveaux traitements. Nous avons en effet une classe consacrée à la gestion des graphes, utilisant deux classes pour les nœuds et les arcs. Le programme principal se trouve dans la « classe *Synchronizer* » qui gère l'ensemble du processus de synchronisation. Toutes les méthodes utiles au traitement des informations dans les graphes ainsi que les techniques de synchronisation existent déjà pour la plupart. Les choix de programmation ont été suffisamment bien faits pour qu'il soit relativement simple de l'étendre à de nouveaux traitements.
- L'ensemble du **code est complet et fonctionnel**.

Cependant, d'un point vue général, le code n'est **pas assez accessible**; bien le comprendre et pouvoir le modifier pose des nombreuses difficultés. C'est pourquoi il nous faut tout d'abord l'améliorer en appliquant les techniques de *refactoring*.

28.2Refactoring

Le refactoring est un procédé qui permet par l'application de plusieurs règles relativement simples de reprendre du code afin d'obtenir les critères d'une "bonne programmation".

Nous recherchons surtout à le rendre plus clair et accessible. Nous avons appliqué ces techniques aux classes "ObjectGraph", "ObjectGraphNode", "ObjectGraphEdge" et "Synchronizer".

Pour ces classes, on obtient ainsi notamment :

- ✓ des commentaires en anglais plus explicites (et avec une meilleure orthographe)
- ✓ plus de commentaires
- ✓ la séparation des étapes de travail
- ✓ la décomposition des méthodes (parfois trop longues)
- ✓ des noms de méthodes ou d'attributs plus explicites
- ✓ un code plus espacé, devenu beaucoup plus lisible

Une fois ce travail réalisé, il devient plus facile pour nous de modifier le code, ainsi que pour les prochains programmeurs qui souhaiteraient apporter leurs améliorations à notre code. Dès ce moment, le travail le plus difficile commence, à savoir analyser le travail à effectuer et l'effectuer, ce qui inclut des modifications à apporter et des extensions à réaliser.

28.3Analyse

Nous avons commencé par étudier la précédente problématique. De cette étude est ressorti le fait que les hypothèses de départ pouvaient être contestées.

En effet, contrairement à ce qui était énoncé, le temps d'un message apparaît sur un séquence de diagramme. L'échelle verticale représente le temps, ainsi on peut lire (ou faire apparaître) le temps d'un message.

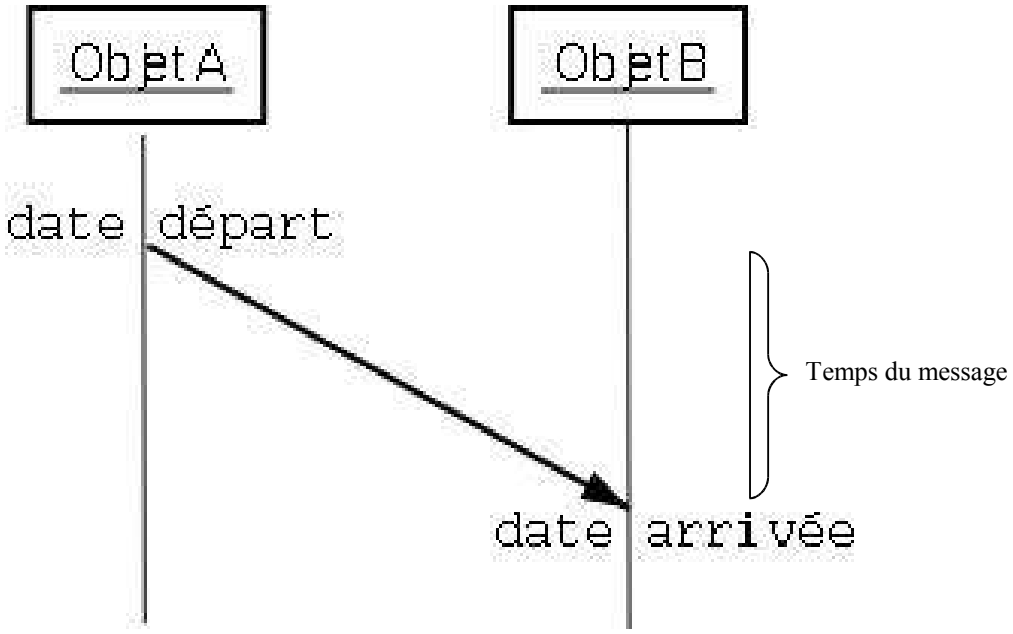


Figure 13: temps d'un message dans un diagramme de séquence

Il n'est pas, également, aussi difficile que çà de gérer du parallélisme dans les diagrammes de séquence en affichant des messages croisés.

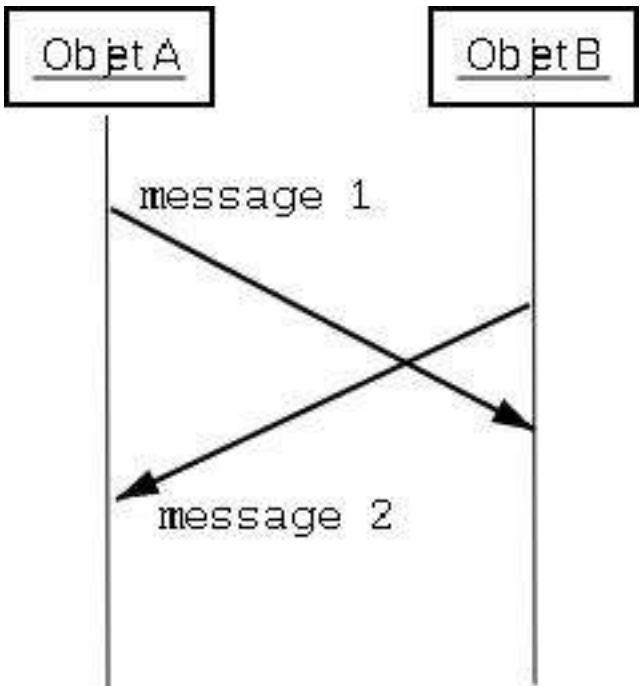


Figure 14: représentation des messages croisés dans un diagramme de séquence

Nous arrivons cependant à une hypothèse de travail commune : le temps d'un message peut être considéré comme nul, mais pour des raisons différentes que nous justifions dans notre problématique.

Cette révision des hypothèses de travail implique quelques bouleversements dans le processus de synchronisation à pondérer par le fait que le code déjà existant est suffisamment complet et bien pensé pour limiter l'étendue des modifications.

Il revient aussi d'intégrer le cas des logs locaux : avec le processus de synchronisation que nous proposons, il devient très simple de gérer ce cas particulier. En effet, quelque soient les types de messages à traiter, nous mettons à jour les horloges de chaque objet; ensuite il ne suffit plus qu'à mettre à jour les dates des messages (par rapport aux horloges des objets émetteurs et receveurs). A priori, il ne devrait pas y avoir de traitement particulier pour les messages locaux.

28.4Problématique

Le temps de transmission d'un message (date arrivée - date départ) entre deux objets est composé du temps réel de parcours du message et du décalage des horloges des objets. Le temps de parcours d'un message est de l'ordre de la milliseconde ou de la seconde, alors que celui du décalage serait plutôt de la minute voire de l'heure. Le premier est donc négligeable devant de second. De plus entre deux objets le temps de parcours du message peut varier à tout moment (en raison du réseau) et est donc imprévisible, on ne peut donc pas se fier à lui pour nos calculs d'estimation. Nous allons donc considérer (comme dans la première version) que ce temps correspond uniquement au décalage des horloges locales.

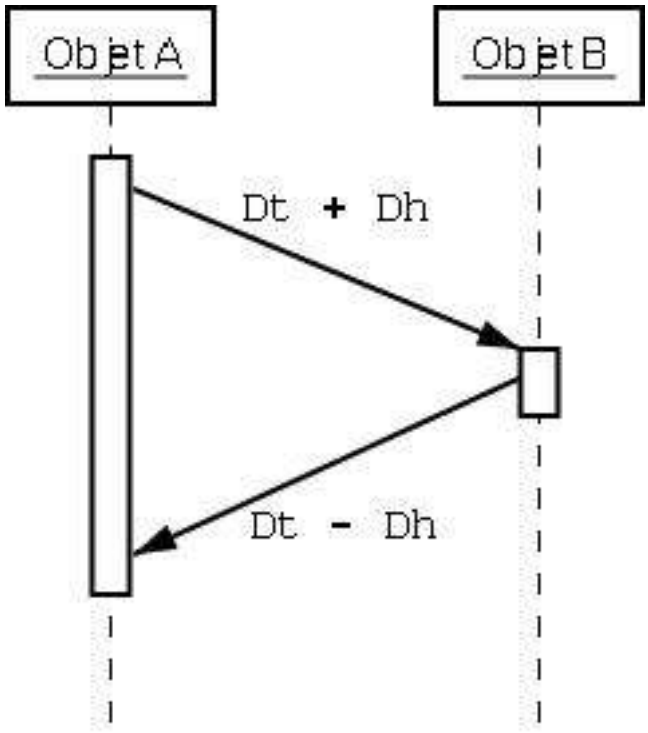


Figure 15: temps effectif idéal (mais non réel) d'un message

Sur ce schéma : « *Dt* » *est le temps réel de transmission du message et* « *Dh* » *le décalage entre les horloges locales des objets A et B*

Le cas présenté dans la figure 16 est utopique, et nous ne considérons donc comme temps du message que le décalage entre les horloges locales des objets A et B.

Compte tenu que nous souhaitons gérer au mieux le parallélisme dans les envoies de messages entre objets distants, il nous faut calculer le décalage d'horloge de chaque objet par rapport à un objet local (toujours comme dans la première version) et appliquer ce décalage aux objets (par une nouvelle technique).

28.5Processus de synchronisation

Comme il est dit précédemment, nous devons synchroniser les horloges locales des objets distants. Pour cela nous allons calculer les décalages d'horloge de chaque objet. Une fois ce décalage calculé, il s'agira de l'appliquer à chaque objet.

Ainsi nous avons juste à mettre à jour les dates d'envoi et de réception des messages en leur appliquant le décalage calculé dans le graphe pour les objets concernés (émetteur et récepteur des messages).

Le processus s'effectue toujours en trois étapes:
- ✓ ajout des messages logués dans le graphe
- ✓ estimation des décalages d'horloge
- ✓ mise à jour des messages

Pour des raisons pratiques, nous conservons une étape de séquentialisation des messages. En effet, le tri des messages par leur date d'émission rend plus pratique leur exploitation et affichage par l'API de visualisation SVG développée dans la nouvelle version de CorbaTrace. Cette étape succède la mise à jour des messages.

28.5.1Ajout des messages logués dans le graphe

Cette étape a été particulièrement bien réussie dans la première version. Nous nous contentons ici de rapidement la détailler afin de bien comprendre l'ensemble du processus de synchronisation.
Pour plus de précisions, il est vivement conseillé de consulter le rapport de l'année précédente.

Lorsque nous avons logué plusieurs messages entre deux objets, on calcule le temps de message minimal *Dmin,* car c'est celui qui approxime le mieux la différence réelle d'horloge entre les objets.Le temps du message est calculé par soustraction de la date de départ de celle d'arrivée. Si le résultat est positif, l'objet émetteur est en retard sur l'objet receveur, alors que s'il est négatif, c'est le contraire (en avance). Logiquement, le décalage des horloges dans l'autre sens (receveur vers émetteur) est le temps opposé à celui déjà calculé.Ainsi on calcule le temps du message et on crée un arc dans le graphe entre les deux objets concernés (deux nœuds). Le poids de cet arc est le temps calculé (décalage entre les deux horloges) et l'arc dans le sens contraire possède un poids opposé au sien. Lorsque l'on a un autre message à ajouter entre les mêmes objets, on calcule son temps et s'il est plus petit en valeur absolue que le précédent (c'est le delta min), alors on le remplace (mise à jour des poids des deux arcs entre les deux nœuds).

28.5.2Estimation des décalages d'horloge

Cette étape est la plus technique du processus. Elle s'inspire de procédés de synchronisation qui existent déjà.

A ce niveau du processus, nous possédons les décalages estimés entre deux nœuds consécutifs. On souhaiterait maintenant calculer tous les décalages par rapport à une horloge commune.

Pour cela, on considère un objet référence, et on calcule tous les décalages des autres horloges locales par rapport à la sienne. On sélectionne un objet référence par composante connexe (ou fortement connexe) dans le graphe. Le choix de cet objet référence n'est pas déterminant puisque quel qu'il soit le choix fait, le résultat final sera en principe le même. Nous avons tout de même garder le choix fait l'année dernière, c'est-à-dire choisir le nœud qui possède le plus d'arcs entrants et sortants comme nœud de référence.

"Le décalage minimal correspond à la somme des arcs du plus long chemin."

Il revient donc de mettre en place dans la classe "`ObjectGraph`" un algorithme de calcul de plus long chemin entre deux nœuds. Ensuite il faudra appliquer cet algorithme à chaque nœud pour connaître son décalage avec le nœud de référence.

Nous avons choisi un algorithme basé sur le modèle de "PERT". Dans ce genre d'algorithme, la difficulté est d'éviter les cycles. (en théorie, le graphe ne doit pas en comporter). Il s'agit d'éviter les circuits de poids infini.

Pour cela nous utilisons les marqueurs mis en place et exploités dans la première version.

Description des étapes de réalisation et de l'algorithme:

- ajout d'un attribut `weightMax` aux nœuds (classe `ObjectGraphNode`) pour stocker la valeur du plus long chemin entre le nœud de référence et le nœud courant.
- on initialise `weightMax` à "moins l'infini" (ou plutôt une forte valeur négative) pour tous les nœuds
- On part du nœud de référence, on fixe son attribut `weightMax` à zéro
- pour chaque voisin, on vérifie que (`weightMax` du nœud courant + le poids de l'arc) < (`weightMax` du voisin), sinon on met ce dernier à jour avec (`weightMax` du nœud courant + le poids de l'arc)
- on marque le nœud courant
- on recommence le traitement avec le voisin s'il n'est pas marqué (ainsi on effectue un parcours en largeur qui garantit de passer par tous les nœuds)
- s'il reste des nœuds non marqués (c'est le cas s'il y a plusieurs composantes connexes) on en sélectionne un comme nœud de référence et on recommence l'opération

Après cela, chaque nœud contient le décalage de son horloge avec celle du nœud de référence.

28.5.3Mise à jour des messages

Tous les décalages sont maintenant calculés dans le graphe, il ne reste plus qu'à mettre à jour les messages. Ceux-ci ont encore des dates erronées à ce niveau du processus de synchronisation. Ce traitement est relativement simple (particulièrement par rapport à l'ancien algorithme qui effectuait nombre de calculs pour un résultat moins satisfaisant dans notre problématique qui a d'autres exigences).

Pour chaque message, on identifie l'objet émetteur, on récupère dans le graphe le décalage calculé et on l'applique à la date d'émission du message. De même pour l'objet receveur du message.

A la suite de cette étape, tous les messages possèdent des dates d'envoi et de réception à jour.

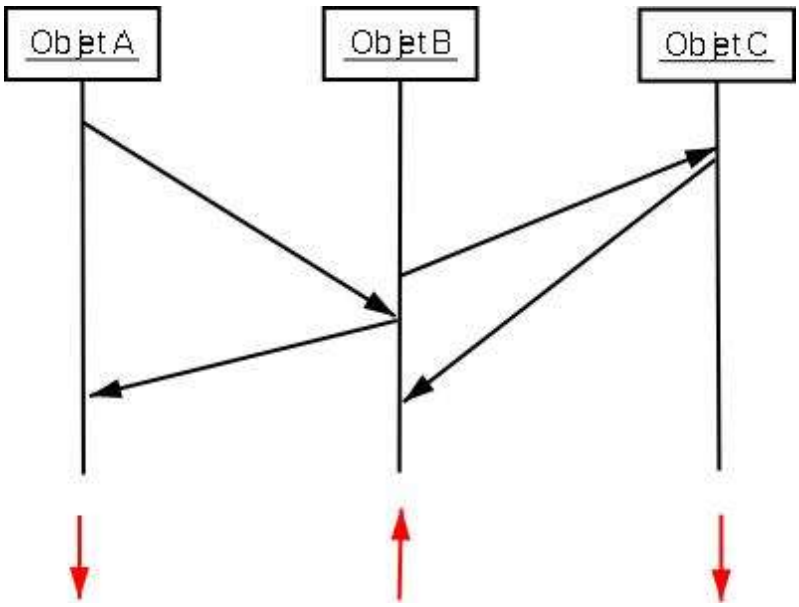


Figure 16: décalages d'horloge à appliquer sur le diagramme de séquence

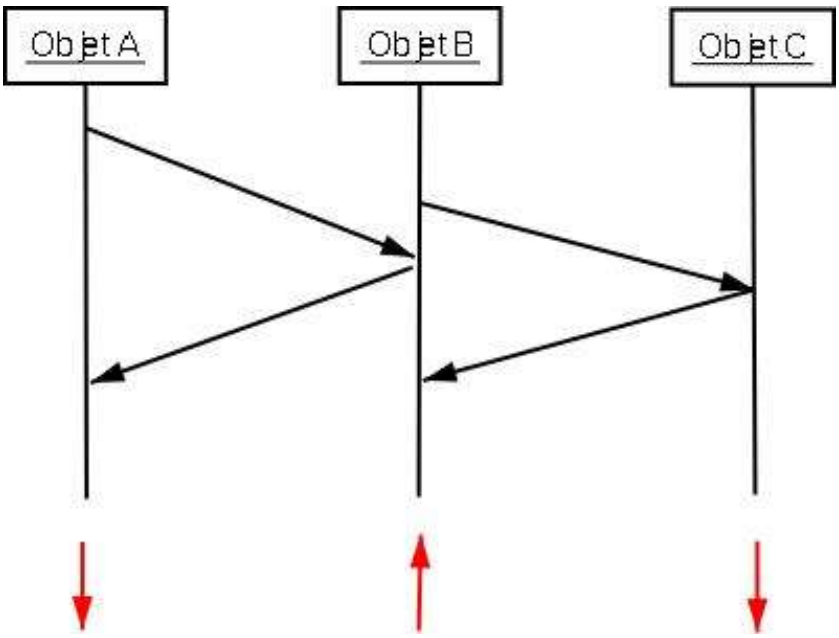


Figure 17: décalages appliqués sur les objets du diagramme de séquence

Les dates des messages vont être mises à jour quelque soientt le type des messages : locaux ou Corba (cf. Figure 16 et Figure 17).

Seul le cas des messages incomplets est plus particulier, car nous ne possédons qu'une seule date (celle d'envoi ou celle de réception). Dans ce cas, on applique la mise à jour de la date uniquement à l'objet connu (émetteur ou récepteur).

28.5.4Limitations

Ce processus, s'il est bien appliqué, garantit une bonne synchronisation des messages. Plus il y a d'échanges de messages et d'informations à traiter, plus le résultat final sera précis et proche de la réalité.

Cependant quelques cas exceptionnels peuvent intervenir et ne seront jamais bien traités par notre synchronisation. Ainsi si des horloges locales différentes avancent à des vitesses différentes, notre algorithme est inefficace car il y aura des incohérences dans les temps des messages. Par exemple, une horloge sera tantôt un retard par rapport à une autre et tantôt en avance par rapport à cette même horloge.

28.5.5Degré d'aboutissement

La synthèse du code est achevée et fonctionnelle. Le code est désormais facile à faire évoluer.

Après avoir effectué de nombreux tests incluant aussi bien les cas standards que les cas particuliers, il s'avère que la synchronisation n'est pas encore pleinement satisfaisante. En effet, un cas important à gérer est l'envoi de messages dans le passé, la cas où la date d'envoie est plus récente que la date de réception en raison d'un décalage des horloges des objets concernés, et ce cas n'est malheureusement pas encore géré.

Toutefois toutes les structures sont présentes pour le gérer et **je m'engage** *(Brice FRANCOIS)* à mettre à jour la synchronisation prochainement.

La synchronisation dans le cadre des cas généraux est cependant satisfaisante. On peut visualiser les échanges de messages dans le temps entre les objets distants.

Pour créer ses propres tests de synchronisation, il suffit de modifier les fichiers XML de logs fusionnés.

29 Interface graphique

L'un des éléments manquants à CorbaTrace pour toucher un plus large public était une interface graphique permettant de réaliser les principales actions du logiciel.

Parmi les actions que nous souhaitions intégrer à l'interface graphique, nous pouvons citer :

- Lancer Log2SequenceDiagram à partir de l'interface après avoir choisi les fichiers de log, le filtre et les options voulus
- Récupérer via le protocole FTP les fichiers de log pouvant se trouver sur des machines autres que la machine sur laquelle l'utilisateur se trouve
- Créer facilement des fichiers de filtre
- Visualiser les diagrammes de séquence générés en SVG

29.1Analyse - Choix technologiques

Le logiciel CorbaTrace étant écrit en langage Java, nous nous sommes tournés pour l'écriture de l'interface graphique vers l'API graphique de ce langage : Swing[12].

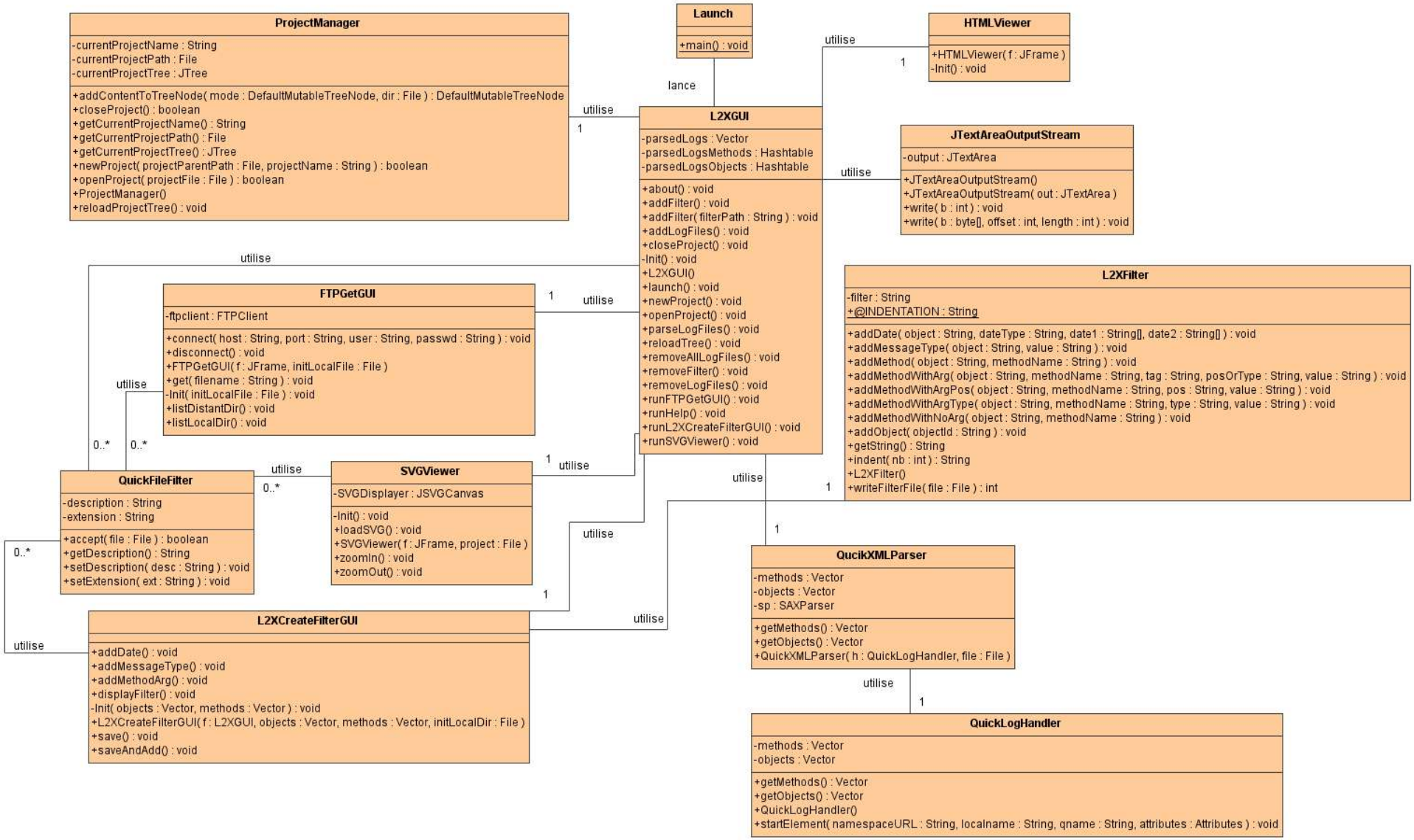
L' Abstract Window Toolkit (AWT) est historiquement la première bibliothèque graphique qui fut proposé par Java. La contrainte de l'AWT est que Java fait appel au système d'exploitation sous-jacent pour afficher les composants graphiques. Pour cette raison, l'affichage de l'interface utilisateur d'un programme peut diverger sensiblement : chaque système d'exploitation dessine à sa manière un bouton.

Or Java se veut être 100% indépendant de la plate-forme utilisée! Pour cette raison, Swing fut donc mis en place pour assurer 100% de portabilité. Ceci à cependant un coup : pour assurer cette portabilité, un composant graphique est dessiner non plus par le système mais par Java, ce qui en terme de temps d'exécution à un prix.

Mais l'avantage de Swing est de proposer des possibilités beaucoup plus étendues et de nombreux widgets graphiques permettant la réalisation d'interfaces complexes de manière relativement aisée.

L'interface graphique est définie dans le package `corbaTrace.gui`, et peut être représenté par le diagramme de classe UML suivant :

(Nb : afin de ne pas surcharger ce diagramme, tous les attributs de classe correspondant aux différents widgets graphiques tels que les boutons ne sont pas présents.)


ProjectManager
-currentProjectName : String
-currentProjectPath : File
-currentProjectTree : JTree
+addContentToTreeNode(mode : DefaultMutableTreeNode, dir : File) : DefaultMutableTreeNode
+closeProject() : boolean
+getCurrentProjectName() : String
+getCurrentProjectPath() : File
+getCurrentProjectTree() : JTree
+newProject(projectParentPath : File, projectName : String) : boolean
+openProject(projectFile : File) : boolean
+ProjectManager()
+reloadProjectTree() : void
Launch
+main() : void
HTMLViewer
+HTMLViewer(f : JFrame)
-Init() : void
L2XGUI
-parsedLogs : Vector
-parsedLogsMethods : Hashtable
-parsedLogsObjects : Hashtable
+about() : void
+addFilter() : void
+addFilter(filterPath : String) : void
+addLogFiles() : void
+closeProject() : void
-Init() : void
+L2XGUI()
+launch() : void
+newProject() : void
+openProject() : void
+parseLogFiles() : void
+reloadTree() : void
+removeAllLogFiles() : void
+removeFilter() : void
+removeLogFiles() : void
+runFTPGetGUI() : void
+runHelp() : void
+runL2XCreateFilterGUI() : void
+runSVGViewer() : void
JTextAreaOutputStream
-output : JTextArea
+JTextAreaOutputStream()
+JTextAreaOutputStream(out : JTextArea)
+write(b : int) : void
+write(b : byte[], offset : int, length : int) : void
L2XFilter
-filter : String
+@INDENTATION : String
+addDate(object : String, dateType : String, date1 : String[], date2 : String[]) : void
+addMessageType(object : String, value : String) : void
+addMethod(object : String, methodName : String) : void
+addMethodWithArg(object : String, methodName : String, tag : String, posOrType : String, value : String) : void
+addMethodWithArgPos(object : String, methodName : String, pos : String, value : String) : void
+addMethodWithArgType(object : String, methodName : String, type : String, value : String) : void
+addMethodWithNoArg(object : String, methodName : String) : void
+addObject(objectId : String) : void
+getString() : String
+indent(nb : int) : String
+L2XFilter()
+writeFilterFile(file : File) : int
FTPGetGUI
-ftpclient : FTPClient
+connect(host : String, port : String, user : String, passwd : String) : void
+disconnect() : void
+FTPGetGUI(f : JFrame, initLocalFile : File)
+get(filename : String) : void
-Init(initLocalFile : File) : void
+listDistantDir() : void
+listLocalDir() : void
QuickFileFilter
-description : String
-extension : String
+accept(file : File) : boolean
+getDescription() : String
+setDescription(desc : String) : void
+setExtension(ext : String) : void
SVGViewer
-SVGDisplayer : JSVGCanvas
-Init() : void
+loadSVG() : void
+SVGViewer(f : JFrame, project : File)
+zoomIn() : void
+zoomOut() : void
L2XCreateFilterGUI
+addDate() : void
+addMessageType() : void
+addMethodArg() : void
+displayFilter() : void
-Init(objects : Vector, methods : Vector) : void
+L2XCreateFilterGUI(f : L2XGUI, objects : Vector, methods : Vector, initLocalDir : File)
+save() : void
+saveAndAdd() : void
QuickXMLParser
-methods : Vector
-objects : Vector
-sp : SAXParser
+getMethods() : Vector
+getObjects() : Vector
+QuickXMLParser(h : QuickLogHandler, file : File)
QuickLogHandler
-methods : Vector
-objects : Vector
+getMethods() : Vector
+getObjects() : Vector
+QuickLogHandler()
+startElement(namespaceURL : String, localname : String, qname : String, attributes : Attributes) : void
utilise
lance
1
0..*

29.1.1Description des classes

Launch : C'est la classe qui lance l'interface graphique principale.

L2XGUI : C'est la classe contenant l'interface graphique principale de Log2SequenceDiagram. C'est à partir de cette interface que l'utilisateur sélectionnera les fichiers de log et les options qu'il veut utiliser, et qu'il pourra lancer le programme Log2SequenceDiagram.
Elle permet également une gestion de projet via une interface graphique en utilisant la classe ProjectManager, de lancer un client FTP de type FTPGetGUI, un visualiseur de fichier SVG de type SVGViewer, et une interface de création de filtre de type L2XCreateFilterGUI.
Pour lancer cette dernière, la classe doit fournir les listes des objets et méthodes présents dans les fichiers de log sélectionnés par l'utilisateur. Par cela, l'interface parse ces fichiers de log en utilisant un QuickXMLParser.

ProjectManager : Cette classe est spécialisée dans la gestion de projet. C'est elle qui gère la création d'un projet (création des répertoires et fichiers) et l'ouverture d'un projet existant.
Elle propose une représentation graphique du projet courant sous l'apparence d'un objet de classe javax.swing.JTree, par l'intermédiaire de la méthode ***getCurrentProjetTree()***. Elle est utilisée par la classe L2XGUI pour la gestion de projet.

JTextAreaOutputStream : Cette petite classe propose un flux de sortie vers un composant Swing JTextArea, en étendant la classe abstraite java.io.OutputStream.
Elle est utilisée dans L2XGUI pour l'affichage des traces de l'exécution de l'application Log2SequenceDiagram dans un composant JTextArea. Ces traces sont écrites dans le programme Log2SequenceDiagram sur la sortie standard System.out, qui est aussi un flux de sortie de type java.io.OutputStream. La classe L2XGUI redirige le flux de sortie System.out dans un flux de type JTextAreaOutputStream le temps de l'exécution de Log2SequenceDiagram afin d'afficher les traces du programme dans l'interface graphique.

QuickLogHandler : Il s'agit d'un handler très simple pour le parsage des fichiers de log par L2XGUI avec la librairie SAX, pour en retirer la liste des objets et des méthodes présentes dans les dits fichiers de log. Ces informations sont utilisées par la classe L2XCreateFilterGUI.

QuickXMLParser : Cette classe définit un parser XML très simple qui utilise un QuickLogHandler pour retirer les objets et les méthodes présents dans les fichiers de log parsés.

HTMLViewer : Il s'agit d'une classe proposant une interface graphique permettant de lire des pages HTML.
Elle est utilisée par L2XGUI pour afficher le fichier d'aide de l'interface graphique qui est au format HTML.

SVGViewer : Cette classe propose une interface graphique permettant d'afficher un fichier graphique au format SVG.
Pour ceci, la classe utilise un composant de type org.apache.batik.swing.JSVGCanvas, issue du projet Batik. Batik, sous projet du projet XML de la fondation Apache, est une technologie basée sur Java permettant de générer et manipuler des fichiers au format SVG. Pour en apprendre plus sur Batik et le format SVG, reportez-vous au chapitre 7.

FTPGetGUI : Cette classe propose une interface graphique pour télécharger des fichiers sur un hôte distant via le protocole FTP. Le protocole FTP permet bien sur bien d'autres possibilités comme ajouter des fichiers à l'hôte distant, y créer des répertoires, ... dans cette classe, seule l'action de récupérer un fichier (GET) est possible, d'où le nom de la classe.

L'utilisation du protocole FTP via le langage Java se fait par l'intermédiaire d'une API développée par Bruce Blackshaw, disponible en licence LGPL sur le site www.entreprisedt.com. Cette API propose l'accès à un client FTP par l'instanciation de la classe `FTPClient`, disposant de fonctions permettant l'accès aux commandes FTP de base.

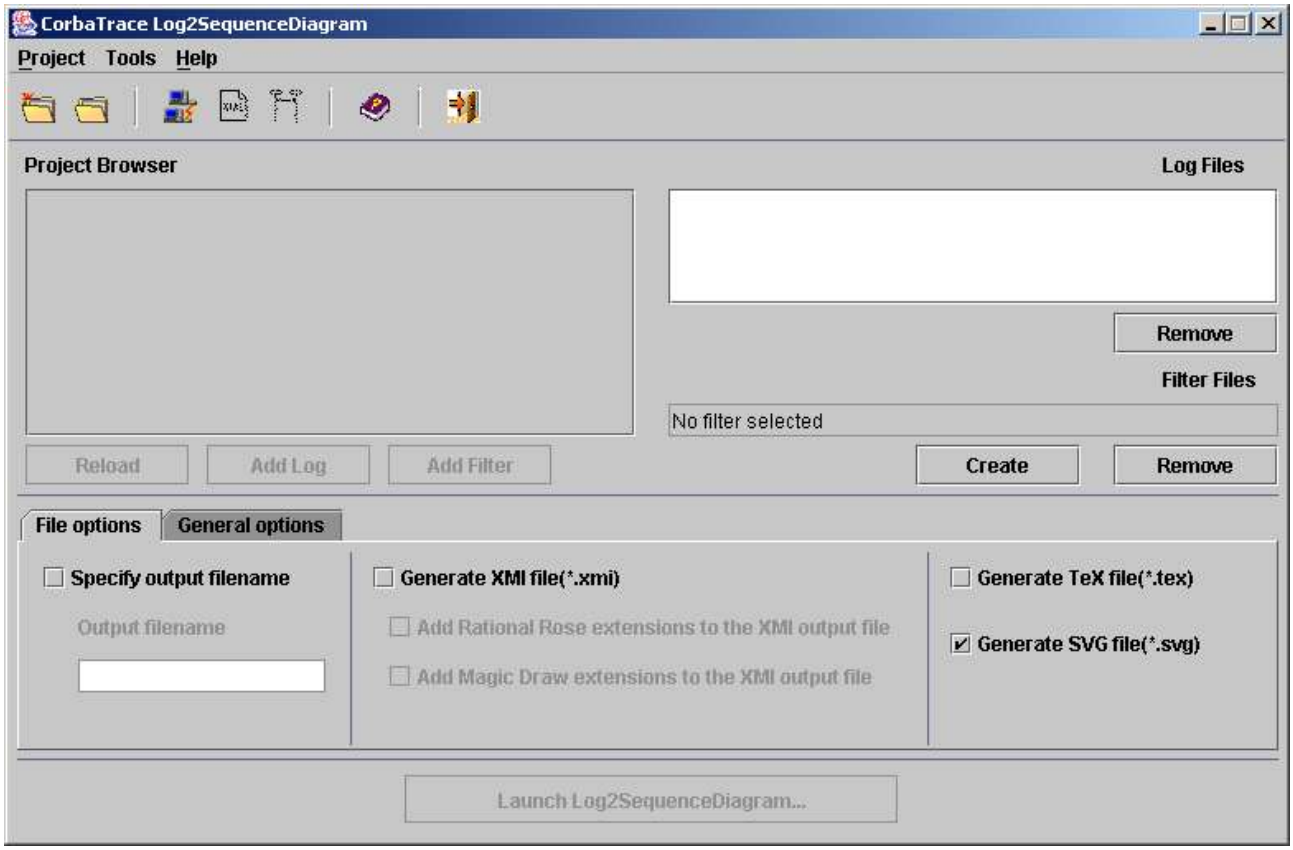
`L2XCreateFilterGUI` : L'interface graphique proposée par cette classe permet, en utilisant la classe `L2XFilter`, la création facile d'un filtre pour l'application `Log2SequenceDiagram`. Dans ce but, l'interface propose dans ces choix les objets et les méthodes qui lui sont passés en paramètre par la classe `L2XGUI` après que celle ci ai parsé les fichiers de log.

`L2XFilter` : Cette classe permet la création d'un fichier de filtre pour l'application `Log2SequenceDiagram` , en proposant des méthodes pour ajouter de manière rapide de nouveaux objets, types de message, dates et méthodes au filtre.

29.2Présentation de l'interface graphique

29.2.1Interface principale

Il s'agit de la première fenêtre que l'utilisateur voit en exécutant le logiciel. Elle propose l'accès à toutes les fonctionnalités.



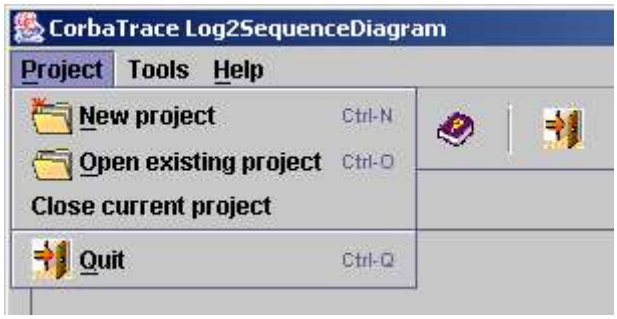
En voici une capture d'écran :

Intéressons nous plus précisément aux différentes parties de cette interface.

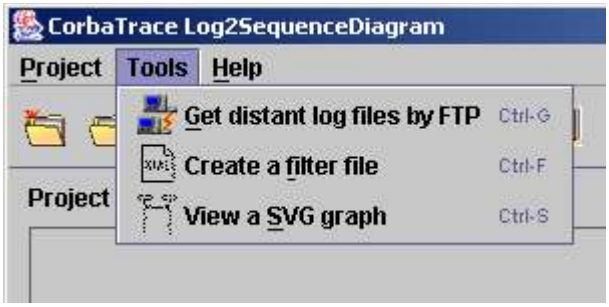
29.2.1.1La barre de menu

Elle est composée de 3 menus :

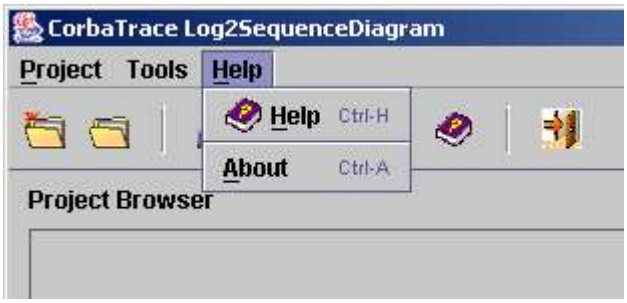
- Le menu **Project** permet de créer un nouveau projet, d'ouvrir un projet existant et de fermer le projet courant. Dans ce menu que se trouve également l'action de quitter l'interface.



- Le menu **Tools** propose un accès aux outils constituant l'interface de Log2SequenceDiagram : Récupérer des fichiers de log distants par FTP, Créer un filtre, Visualiser un fichier SVG



- Le menu **Help** donne accès à l'aide du logiciel ainsi qu'a la fenêtre About contenant le nom des auteurs de CorbaTrace et l'information concernant la licence du logiciel.



29.2.1.2 La barre d'outils

Elle propose par l'intermédiaire d'icônes un accès aux principales fonctions de l'interface, dans l'ordre :

- Créer un objet
- Ouvrir un objet
- Récupérer des fichiers de log distants par FTP
- Créer un filtre
- Visualiser un fichier SVG
- Accéder à l'aide
- Quitter l'interface

29.2.1.3L'explorateur de projet

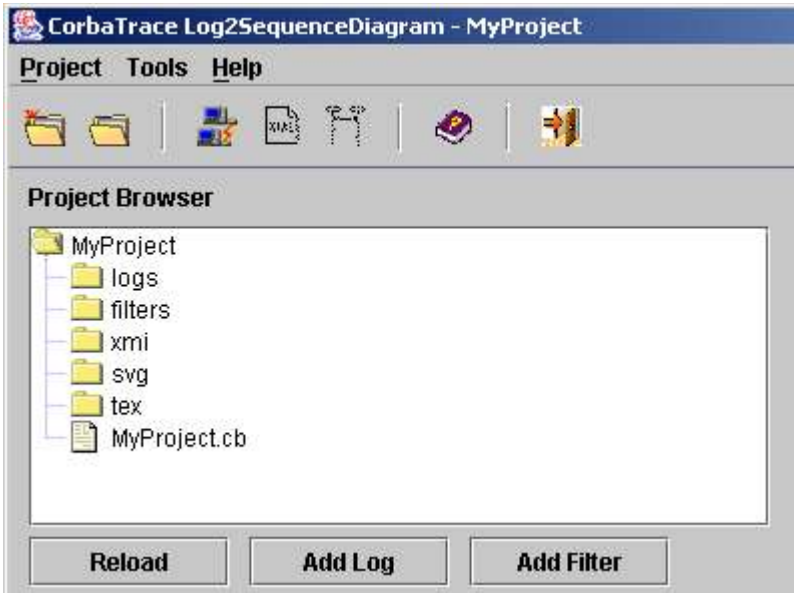
Le concept de projet dans l'interface graphique de Log2SequenceDiagram correspond à un ensemble de répertoires permettant à l'utilisateur d'ordonner son travail dans l'interface.

Lorsqu'un projet CorbaTrace est créé, les répertoires et fichiers suivants sont effectivement créés sur le disque :

<nomDuProjet> (Répertoire racine du projet, portant le nom de celui-ci)
|_ **filters**
(Répertoire dans lequel l'utilisateur est invité à mettre les fichiers de filtre)
|_ **logs**
(Répertoire dans lequel l'utilisateur est invité à mettre les fichiers de log)
|_ **svg**
(Répertoire dans lequel seront enregistrer les fichiers SVH générés)
|_ **tex**
(Répertoire dans lequel seront enregistrer les fichiers TEX générés)
|_ **xmi**
(Répertoire dans lequel seront enregistrer les fichiers XMI générés)
|_**<nomDuProjet>.cb**
(Fichier projet CorbaTrace, permettant d'ouvrir le projet)

L'explorateur de projet permet ensuite de naviguer entre ces répertoires, et d'effectuer des actions sur les fichiers qu'ils contiennent.

Voici l'explorateur de projet après la création d'un projet nommé MyProject :



Pour l'instant les répertoires ne peuvent être ouverts car ils ne contiennent aucun fichier. On remarque que le nom du fichier s'est inscrit dans la barre de titre.

Après avoir ajouté deux fichiers de log dans le répertoire logs et un fichier de filtre dans le répertoire filters, voilà a quoi ressemble l'explorateur de fichier :

Trois boutons se trouvent sous l'explorateur de projet.

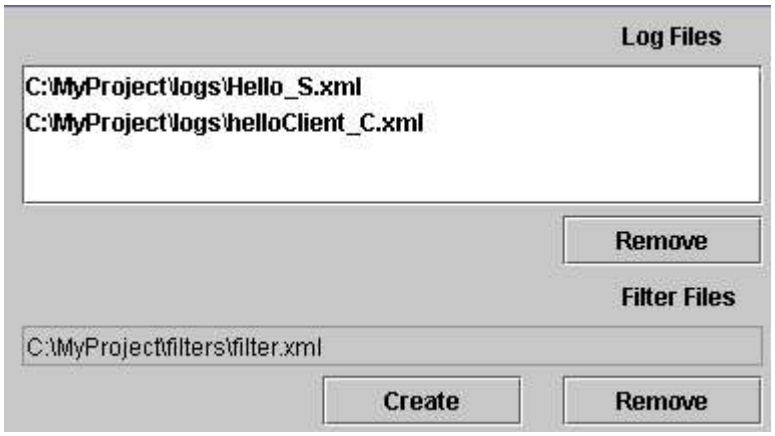
Le bouton **Reload** permet de rafraîchir l'affichage de l'explorateur de projet.

Le bouton **Add Log** permet d'ajouter à la liste des fichiers de log à traiter les fichiers sélectionnés dans l'explorateur.

Le bouton **Add Filter** permet de sélectionner comme filtre à appliquer le fichier sélectionné dans l'explorateur.

29.2.1.4 Les fichiers de log et de filtre

A gauche de l'explorateur de projet se trouve la liste des fichiers de log et le fichier de filtre sélectionnés par l'utilisateur pour être utilisés par l'application `Log2SequenceDiagram`.



Le bouton **Remove** de la partie **Log File**s permet de supprimer les fichiers de log sélectionnés de la liste.

Le bouton **Create** de la partie **Filter Files** permet de lancer l'interface de création de filtre.

Le bouton **Remove** de la partie **Filter Files** permet de supprimer le fichier de filtre de la sélection.

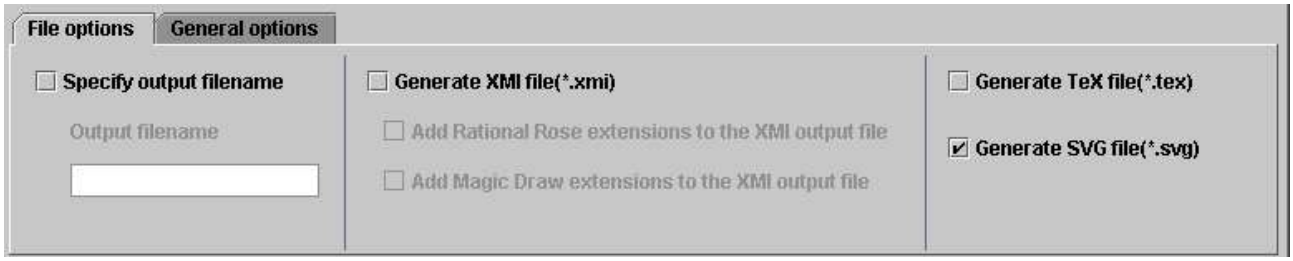
29.2.1.5 Les options

Les options de `Log2SequenceDiagram` sont séparées en deux parties :

- les options du fichier de sortie
- les options dites 'générales'

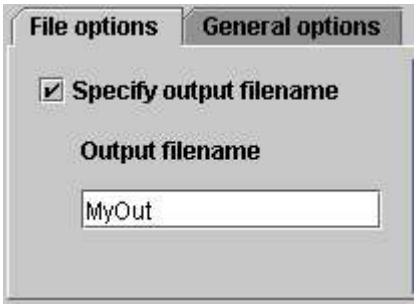
29.2.1.5.1Options du fichier de sortie

Elles permettent de configurer quel sorte de fichier(s) seront généré(s) et le nom du ou des fichier(s).



La partie de gauche permet de spécifier le nom que l'on souhaite pour le ou les fichier(s) de sortie. Par défaut, il s'agit de **out**.

Lorsque la case à cocher **Specify output filename** est sélectionnée, le champ de texte devient accessible et l'utilisateur peut rentrer le nom qu'il souhaite.



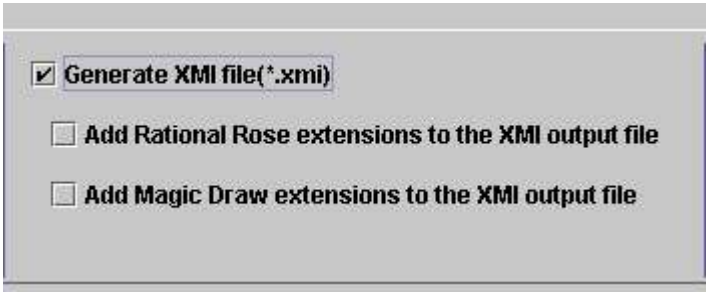
Les autres options concernent les types du ou des fichiers générés.

Dans la partie centrale, l'utilisateur peut choisir de généré le diagramme de séquence sous la forme d'un fichier XMI. XMI (XML Metadata Interchange) est une spécification de l'OMG, basé sur XML, qui permet l'échange de données méta entre des outils de modélisation basés sur les spécifications UML.

Le fichier XMI sera enregistré dans le répertoire **xmi** du projet.

Chaque AGL utilise dans les fichiers XMI des extensions qui lui sont propres pour la représentation graphique. La décision avait été prise par l'équipe de développement précédente de proposer d'ajouter au fichier XMI généré les extensions propres aux logiciels Rational Rose et Magic Draw.

Ainsi lorsque choisi l'option de générer un fichier XMI, les cases à cocher lui permettant d'ajouter s'il le souhaite les extensions Rational Rose ou Magic Draw deviennent accessibles.



L'utilisateur peut également choisir de générer un diagramme de séquence sous forme de fichier Latex.

Le fichier Latex sera enregistré dans le répertoire **tex** du projet.

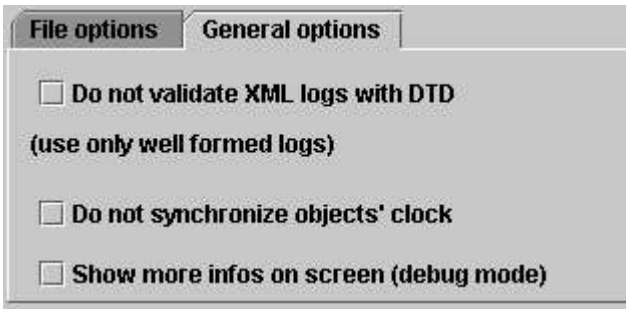
Enfin le dernier format proposé pour le diagramme de séquence généré est le format SVG[7].

Le fichier SVG sera enregistré dans le répertoire **svg** du projet.

Le format SVG est celui que nous conseillons aux utilisateurs car CorbaTrace propose son propre visualiseur SVG intégré. Pour marquer ce conseil, la case à cocher sélectionnant la génération d'un fichier SVG est sélectionné par défaut.

29.2.1.5.2Options générales

Ces options sont celles qui ne concernent pas le fichier de sortie.



L'utilisateur peut d'abord choisir de ne pas valider les fichiers de log avec leur DTD (lors du parsage).

La deuxième option propose à l'utilisateur de ne pas synchroniser les horloges des différents objets.

Enfin, l'utilisateur peut choisir d'afficher plus d'informations dans les traces de sorties, cette option étant essentiellement utilisée à des fins de debuggage.

Lorsque l'utilisateur a sélectionné ses fichiers de log, éventuellement son filtre, ses options, il peut lancer la génération du diagramme de séquence en cliquant sur le bouton **Launch Log2SequenceDiagram**.

29.2.2 Les outils

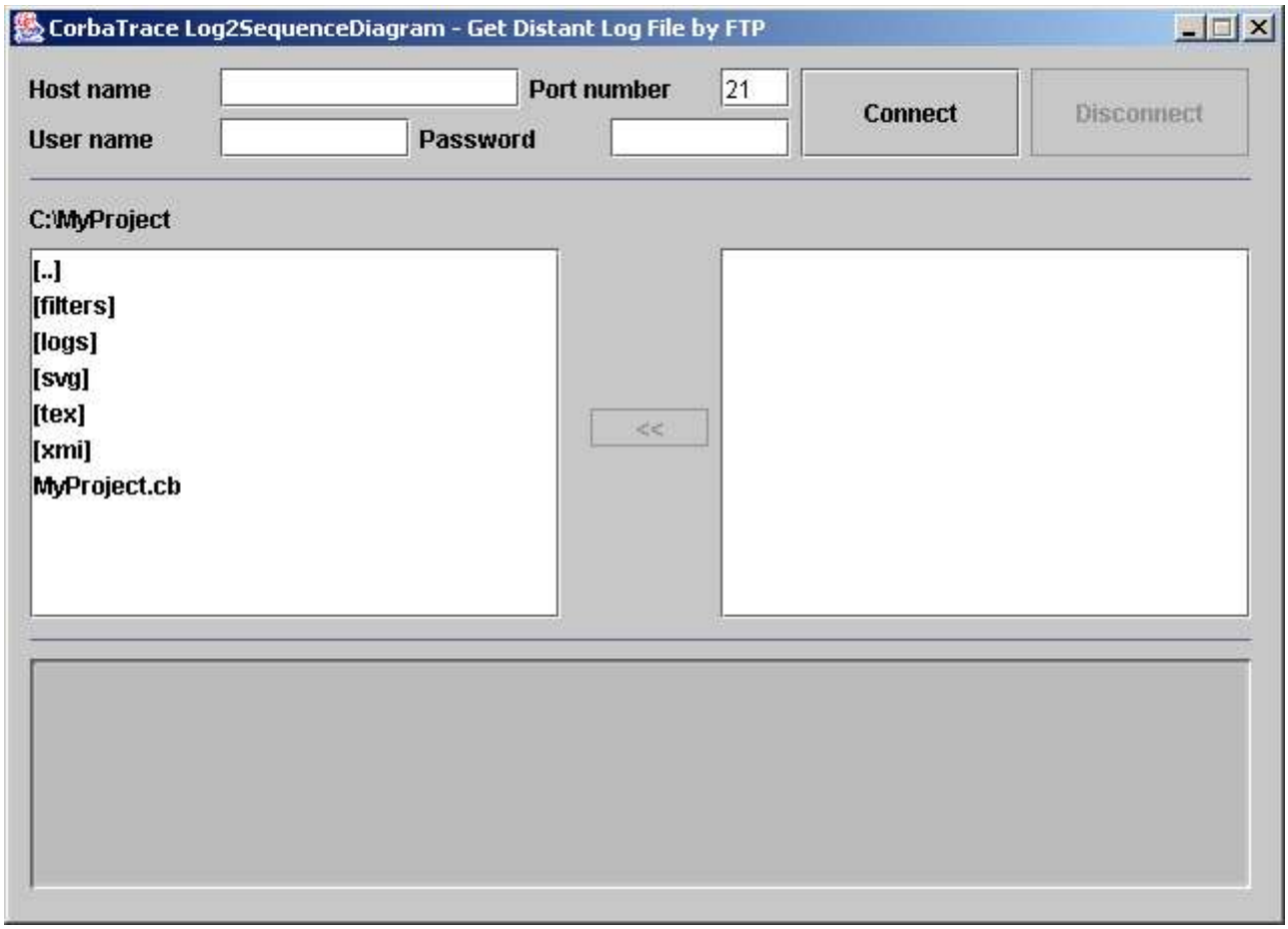
L'interface graphique propose 3 outils facilitant l'utilisation de Log2SequenceDiagram.

29.2.2.1Récupérer des fichiers de log distants par FTP

CorbaTrace est un outil d'observation pour application répartie utilisant CORBA. Qui dit répartie dit que les objets loggés peuvent se situer sur des machines distantes de celle sur laquelle l'utilisateur utilise Log2SequenceDiagram. Il convenait donc de lui fournir un outil lui permettant de récupérer les fichiers de log éparpillés sur des machines distantes.

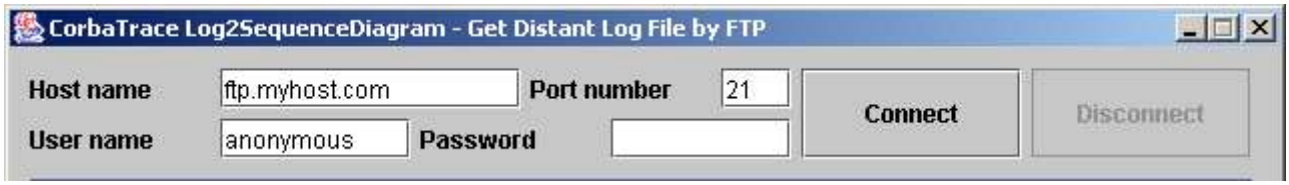
Le protocole standard pour le transfert de fichier est le protocole FTP (File Transfert Protocol). C'est celui qui a été choisi pour l'outil FTPGetGUI.

Notre outil propose donc une interface graphique permettant de connecter à un hôte distant et de récupérer des fichiers sur cet hôte via le protocole FTP.



29.2.2.1.1Connexion à l'hôte

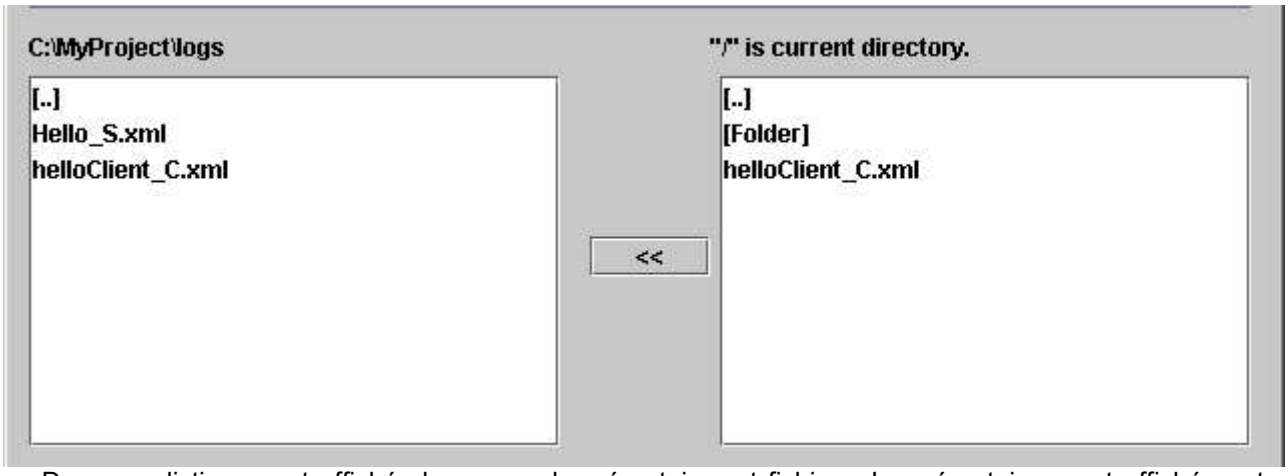
Pour se connecter à un hôte distant, l'utilisateur fournit le nom ou l'adresse IP de cette machine, le port FTP de celle-ci (le port utilisé généralement pour le FTP est le port 21), et éventuellement son nom d'utilisateur et son mot de passe nécessaire pour sa connexion, puis clique sur le bouton **Connect**.



Une fois connecté, le bouton Disconnect est accessible à l'utilisateur pour le déconnecter de l'hôte distant.

29.2.2.1.2Explorateurs

La partie Explorateurs de l'interface graphique représente les listings des répertoires local, et distant une fois connecté à l'hôte.



Dans ces listings sont affichés les noms des répertoires et fichiers. Les répertoires sont affichés entre crochets, de cette manière : **[répertoire]**.

A gauche se trouve le listing du répertoire local. Si un projet est ouvert, le répertoire local est le répertoire du projet. Sinon, il s'agit du répertoire d'exécution de l'interface graphique.

A droite se trouve le listing du répertoire distant, une fois la connexion à l'hôte établie.

Au milieu, un bouton **<<** devient accessible une fois la connexion à l'hôte établie. Il permet de télécharger un fichier, situé sur l'hôte, sur la machine locale.

29.2.2.1.3Traces

Enfin, en bas, se trouve un écran où s'écrivent les traces de la connexion et des échanges avec l'hôte lors de la connexion FTP.

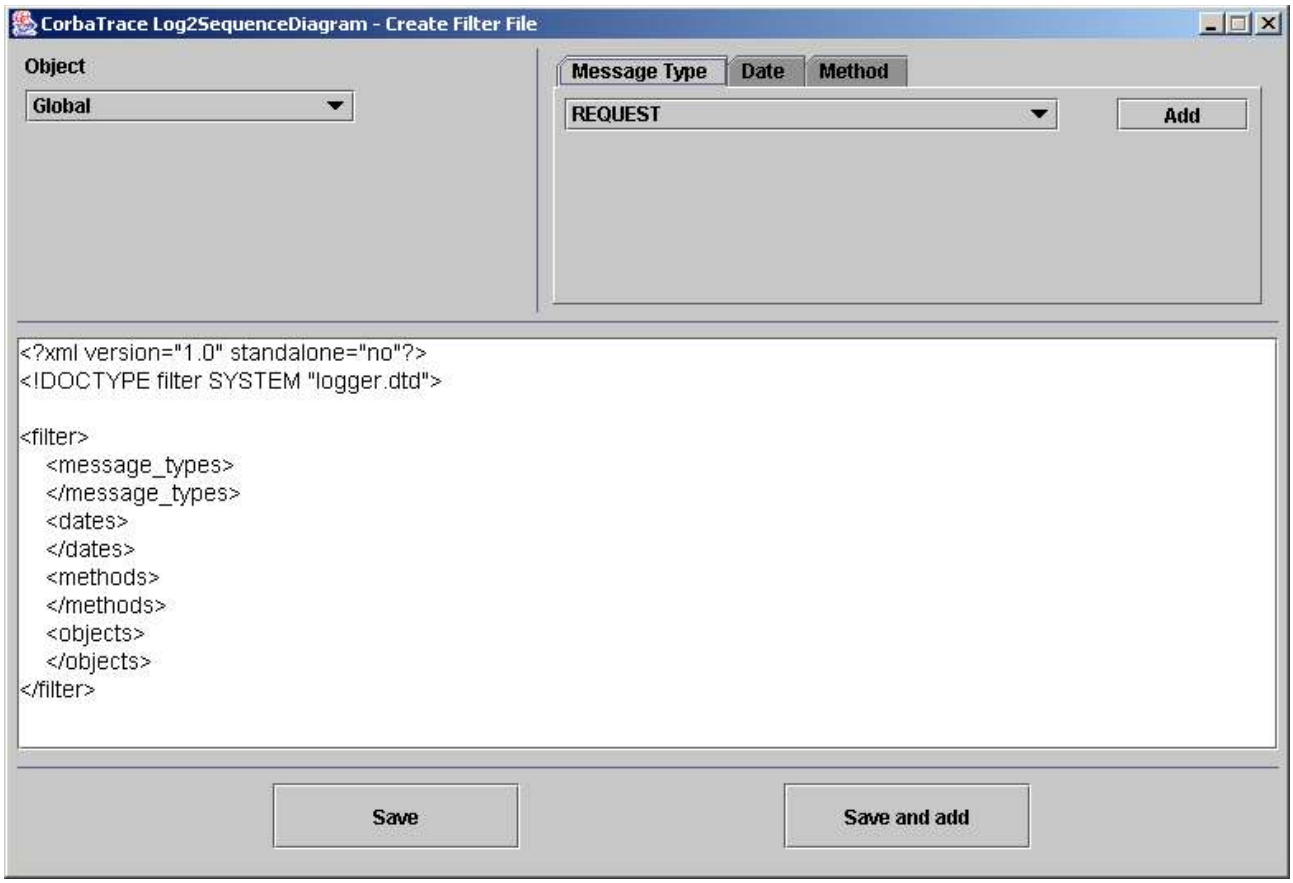
29.2.2.2 Créer un filtre

Le système de filtre est une fonctionnalité quasiment indispensable de l'outil Log2SequenceDiagram. Sans filtre, avec des fichiers de log pouvant contenir des centaines de messages, l'exploitation des données peut s'avérer très difficile. Les filtres permettent de mieux cibler les messages que l'utilisateur souhaite visualiser.

Un filtre est un fichier XML indiquant avec quelles informations filtrer les fichiers de log. Quatre types d'informations sont filtrables :

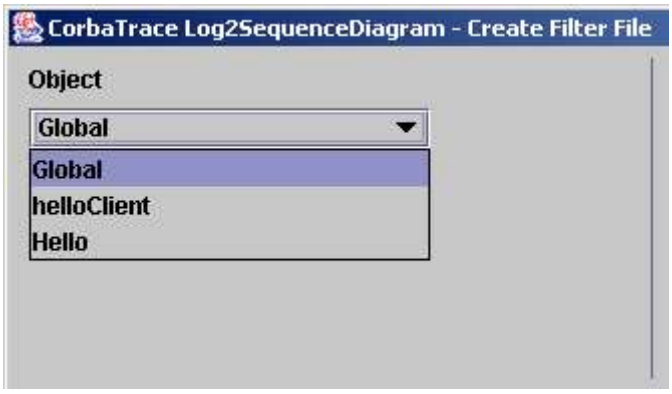
- les types de messages (REQUEST, REPLY, EXCEPTION, BROKEN_REQUEST, BROKEN_REPLY, BROKEN_EXCEPTION)
- les objets (par leur identifiant)
- les dates (après, avant une date donnée, entre deux dates données)
- les opérations (par leur nom et par leurs arguments)

Notre outil de création de filtre facilite la création d'un tel fichier en proposant à l'utilisateur de simplement remplir des champs de texte au lieu d'avoir à éditer le fichier XML de filtre.



29.2.2.2.1L'objet

Une information s'applique sur un objet. Cette liste déroulante propose ainsi à l'utilisateur de choisir l'objet sur lequel appliquer l'information de filtre qu'il a créé.



Cette liste déroulante est remplie avec les objets contenus dans les fichiers de log que l'utilisateur a placé dans sa liste de fichiers de log.

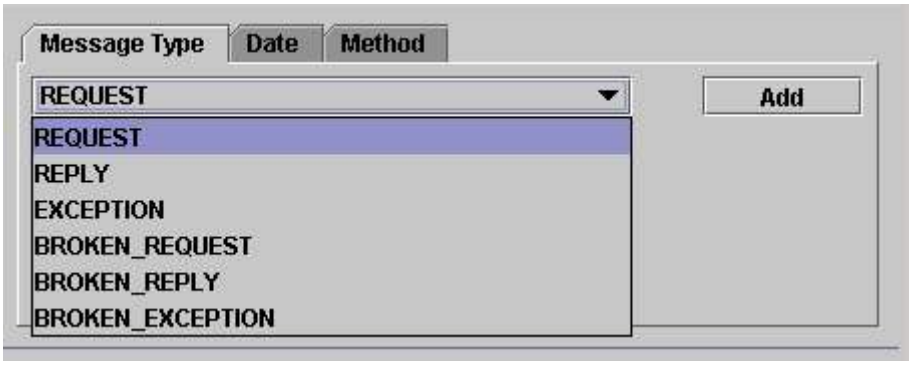
Avec ses objets se trouve un objet **Global** qui permet d'appliquer l'information de filtre sur tous les objets.

29.2.2.2.2L'information de filtre

L'utilisateur va ici choisir très facilement l'information avec laquelle il veut filtrer les fichiers de log. Trois onglets lui sont proposés selon l'information qui l'intéresse.

29.2.2.2.2.1Le type de message

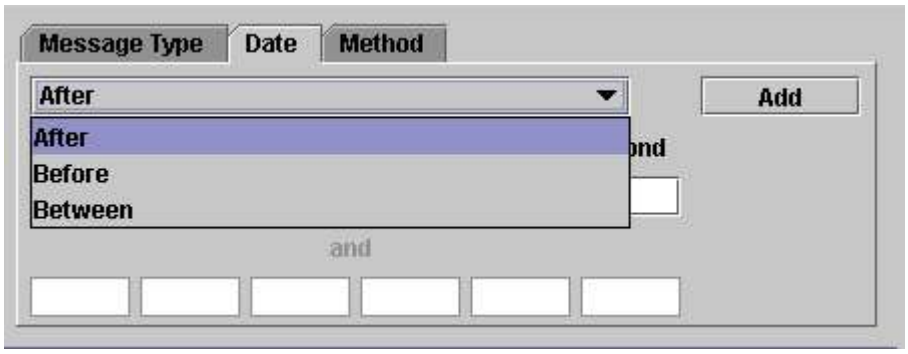
L'utilisateur peut choisir quel type de message il veut filtrer.



En cliquant sur **Add**, l'information de filtre du type de message sur l'objet sélectionné se rajoute dans le fichier de filtre.

29.2.2.2.2.2La date

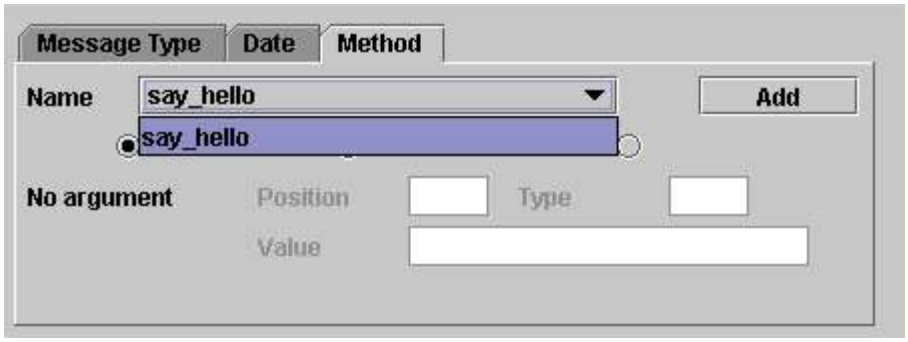
L'utilisateur peut choisir de filtrer selon la date. Dans la liste déroulante, il peut choisir de filtrer avant, après ou entre deux dates.



Puis, après avoir rentré une ou deux dates (selon le type de filtre d'information de filtre choisi), l'utilisateur ajoute cette information de filtre sur l'objet sélectionné en cliquant sur **Add**.

29.2.2.2.2.3Les opérations

Enfin, le dernier type d'information sur lequel l'utilisateur peut filtrer est les opérations.

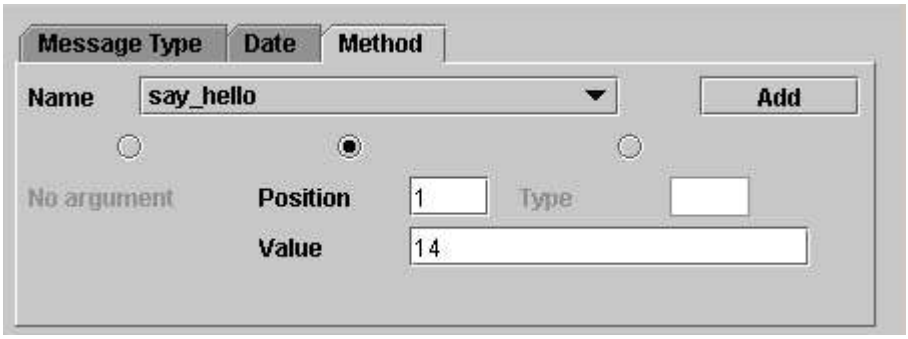


La liste déroulante contient les opérations contenues dans les fichiers de log que l'utilisateur a placé dans sa liste de fichiers de log.

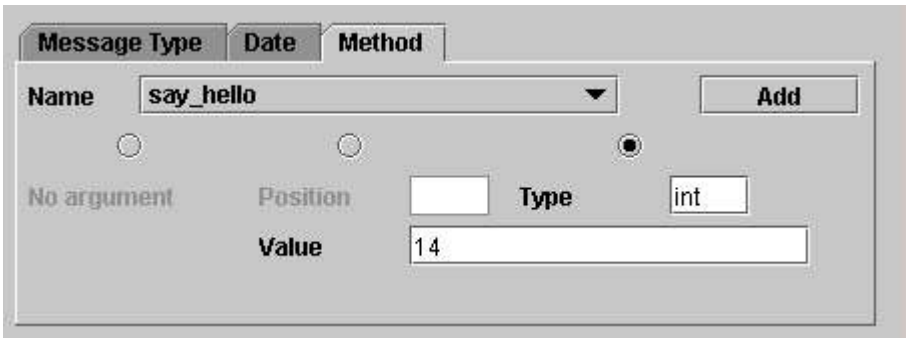
Après avoir choisi l'opération voulue, l'utilisateur a le choix entre trois types de filtre pour celle-ci.

Le premier est de filtrer seulement sur le nom de la méthode, sans tenir compte des arguments.

Le second est de filtrer sur un argument situé à une position (entre 1 et n) associé à une valeur.

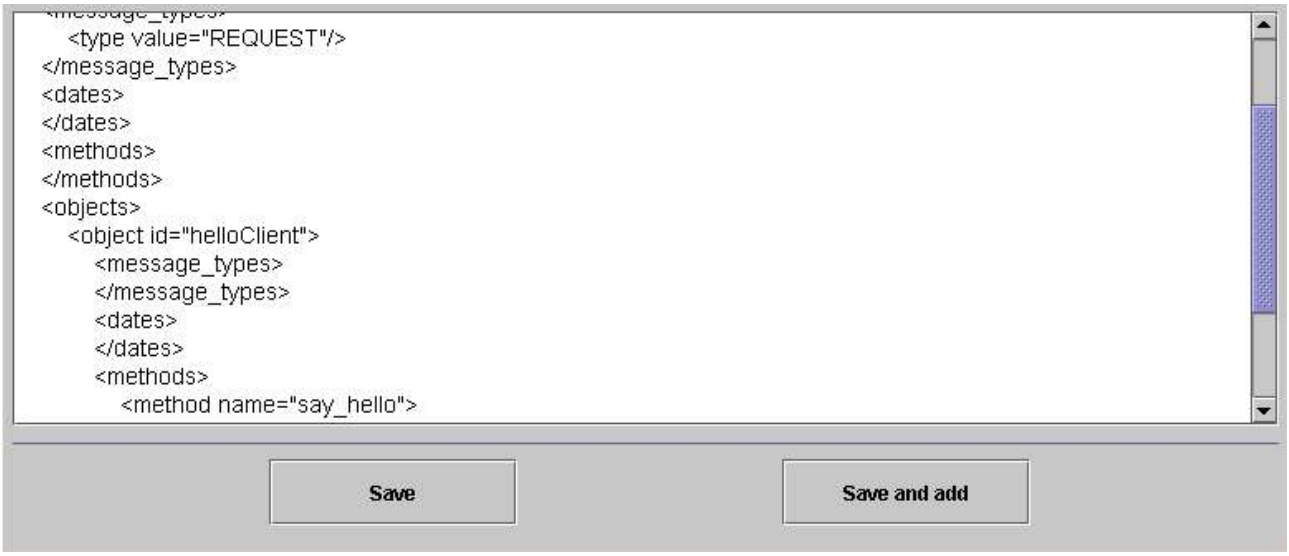


Le troisième est de filtrer sur un argument selon son type associé à une valeur.



29.2.2.2.3Le filtre

Une zone de texte permet de visualiser le filtre au fur et à mesure de sa construction par l'utilisateur.



Deux boutons se trouvent sous cette zone de texte.

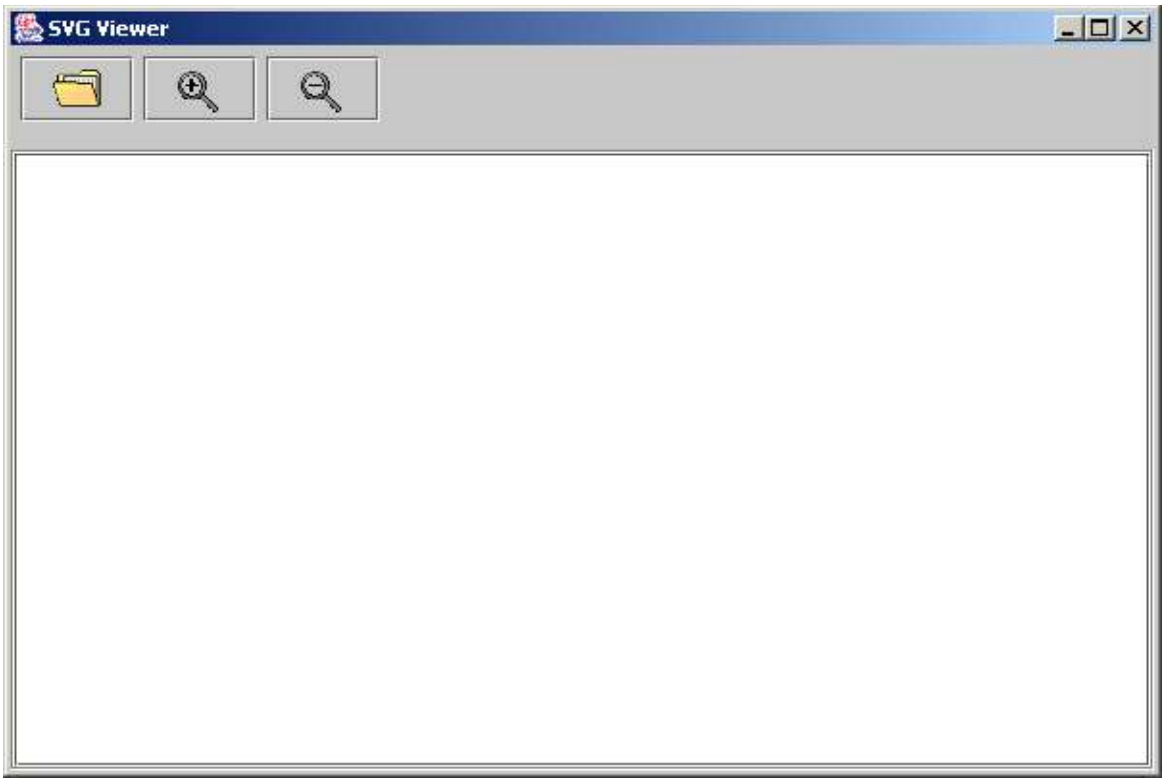
Le bouton **Save** permet de sauvegarder le filtre dans un fichier.

Le bouton **Save and add** permet de sauvegarder le filtre dans un fichier, puis de sélectionner ce filtre comme fichier de filtre à appliquer.

29.2.2.3 Visualiseur SVG

Log2SequenceDiagram propose la génération de diagramme de séquence au format SVG. Ce format de fichier graphique, s'appuyant sur XML, peut être lu avec par exemple un navigateur Internet en lui associant un plug-in spécifique.

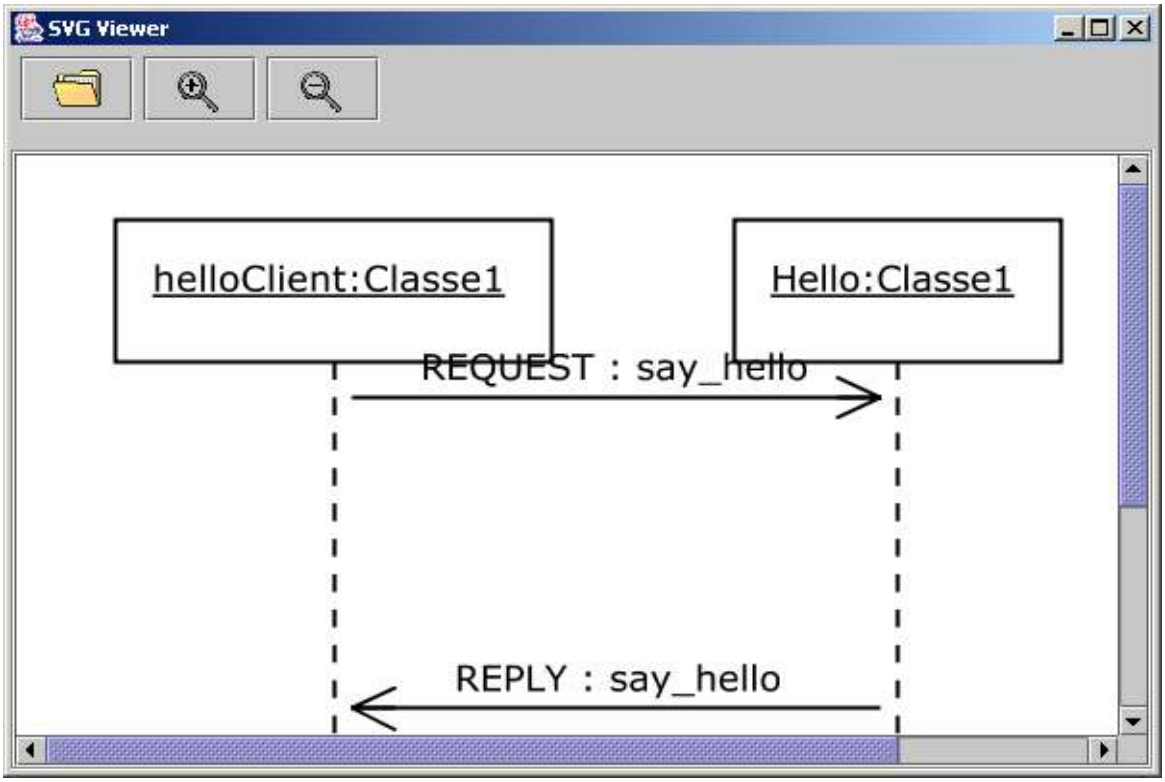
Afin que l'utilisateur puisse visualiser ses diagrammes de séquence sans n'avoir besoin de rien d'autre que l'interface graphique de l'application Log2SequenceDiagram, un visualiseur de SVG a été intégré à elle-ci.



La base d'outils permet les actions de base.

Le premier bouton permet d'ouvrir un fichier SVG.

Le deuxième bouton est utilisé pour faire un zoom avant dans le document SVG, tandis que le troisième permet de faire lui un zoom arrière.



Une fois le document SVG ouvert, plusieurs actions sont accessibles par des combinaisons de touche clavier et de touche souris.

En voici la liste :

- SHIFT + clic gauche : déplacer l'image
- SHIFT + clic droit : zoom manuel
- CTRL + clic gauche : zoom sur une zone
- CTRL + clic droit : rotation de l'image

30 Diagrammes de séquence et SVG

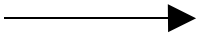
30.1 UML et les diagrammes de séquence

Définition:

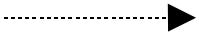
Les diagrammes de séquence font parti du langage UML (Unified Modeling Langage) défini par l'OMG [6]. Ils permettent de représenter des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages. L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe. La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

Type de message:

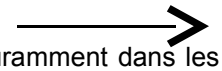
Comme vous pouvez le voir dans la figure 1, UML propose un certain nombre de stéréotypes graphiques pour décrire la nature du message. Nous définissons ici ceux que nous employons pour le projet.

- Message d'appel de méthode (ou de procédure) : 

Ce message permet à un objet d'appeler une méthode à un autre objet. Il montre aussi la création d'un objet à partir d'un autre objet comme l'indique la figure 1.

- Message de retour d'appel de méthode : 

Ce message est la réponse de l'objet appelé.

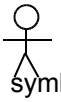
- Message simple : 

C'est celui utilisé le plus couramment dans les diagrammes de séquences. Il dénote l'envoi d'un message classique d'interaction sans appel à une méthode.

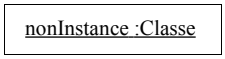
Remarque : Dans un diagramme de séquence, les flèches sont une indication sur le niveau de détails du diagramme. Il est tout à fait possible de représenter un diagramme de séquence symbolisant des interactions d'appel de méthodes en mettant uniquement des messages simples.

Type d'objets:

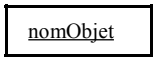
Les stéréotypes graphiques existent également pour les objets interagissant :

- Acteur : 

  Les Acteurs symbolisent généralement l'individu humain interagissant avec le système tout comme dans les diagrammes de cas d'utilisation.

- Instance d'une classe : 

  Lorsque l'on veut symboliser l'instance d'une classe, on doit mettre son nom puis la classe qu'elle instancie séparée par deux points à l'intérieur d'un rectangle.

- Objet Concurrent : 

  L'objet concurrent représente un objet interagissant avec le système. On ne précise rien sur la nature de l'objet (instance ou acteur) sachant qu'il peut être instance ou acteur.

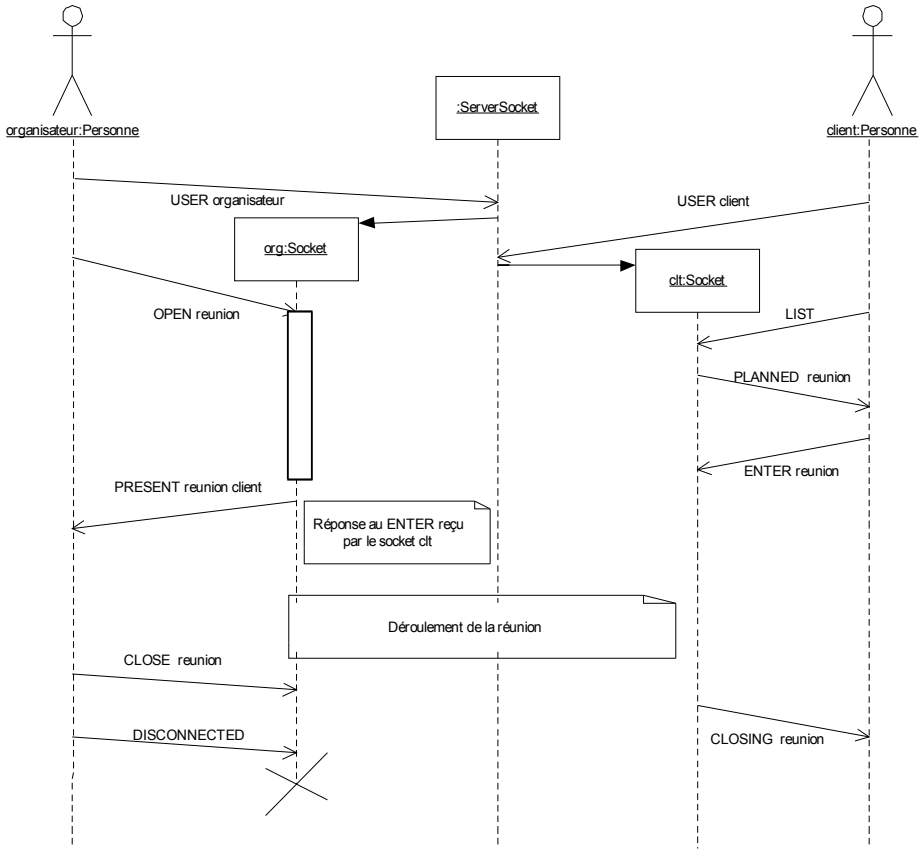


Figure 18: Exemple de diagramme de séquence

Autres symboles:

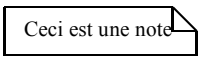
Pour préciser la sémantique du diagramme de séquence, l'OMG a conçu quelques symboles utiles. Nous en présentons quelques-uns que nous utilisons:

- Action : 

Les actions se situent sur l'axe de temps d'un objet. Il symbolise l'exécution d'une action en un temps allant du début du rectangle jusqu'à la fin.

- Croix :

La croix indique la fin de vie de l'objet.

- Note : 

Elle permet d'ajouter des informations au diagramme de séquence pour le rendre plus explicite. Elle peut aussi mettre l'accent sur un point du diagramme.

Avantages du diagramme de séquence:

Le diagramme de séquence est facile à comprendre. Il peut définir de façon détaillée les interactions entre objets. Il est parfaitement adapté au projet. L'utilisateur à l'aide de ce type de schéma peut sans difficulté visualiser les interactions de ses applications Corba.

30.2 Choix de SVG

30.2.1Présentation de SVG

La nécessité d'avoir des graphiques vectoriels (pour économiser de la bande passante) et re-taillables (pour pouvoir facilement les placer dans une mise en page et pour pouvoir les zoomer) a conduit le W3 Consortium [7] a créé un Groupe de travail en 1998. Le W3C est un organisme qui créer des standards pour le Web. Il rassemble des représentants de Microsoft, Autodesk, Adobe, IBM, Sun, Netscape, Xerox, Apple, Corel, HP,ILOG entre autres.

SVG est un langage de description de graphiques 2D en XML Il permet trois types d'objets graphiques : des *formes* vectorielles (traits, courbes, ..), des images et du texte. Les objets graphiques peuvent être groupés, transformés, composés dans d'autres objets et recevoir des attributs de style. Les graphiques SVG sont interactifs et dynamiques. Leur animation peut être définie soit à l'intérieur des fichiers SVG soit dans un langage de script externe. SVG possède une interface avec DOM pour que l'animation puisse accéder à tous les éléments et les attributs.

30.2.2Exemple

Le code suivant permet de créer la figure 1.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
<!--création de la fenêtre de taille 500 sur 500-->
<svg width="500" height="500" >
<!—La balise g permet de regrouper les objets pour leurs attribuer des attributs
communs-->
<!—ici le text-rendering et le shape-rendering-->
<g style="text-rendering:optimizeLegibility;shape-rendering:default">
<!--création du texte SVG Demo : Basic SVG shapes-->
    <text x="5" y="20" style="font-size:22">SVG Demo: Basic SVG shapes</text>
    <g style="stroke:black; fill:none; shape-rendering:default" >
     <!--création du cercle-->
      <circle cx="70"  cy="100" r="50"  />
     <!--création du rectangle-->
      <rect x="150" y="50" width="135" height="100" />
      <!--création des lignes-->
      <line x1="325" y1="150" x2="375" y2="50" />
      <line x1="375" y1="50"  x2="425" y2="150" />
      <!--création du polyligne-->
      <polyline points="50,250,75,350,100,250,125,350,150,250,175,350" />
     <!--création du polygone-->
      <polygon points="250,250,297,284,279,340,220,340,202,284,250,250" />
      <!--création de l'ellipse-->
      <ellipse cx="400" cy="300" rx="72" ry="50" />
    </g>
    <!--création des textes sous les formes-->
    <g style="text-rendering:optimizeSpeed">
      <text x="50"  y="175">Circle</text>
      <text x="175" y="175">Rectangle</text>
      <text x="355" y="175">Lines</text>
      <text x="50"  y="375">Polyline</text>
      <text x="225" y="375">Polygon</text>
      <text x="375" y="375">Ellipse</text>
    </g>
  </g>
</svg>
```

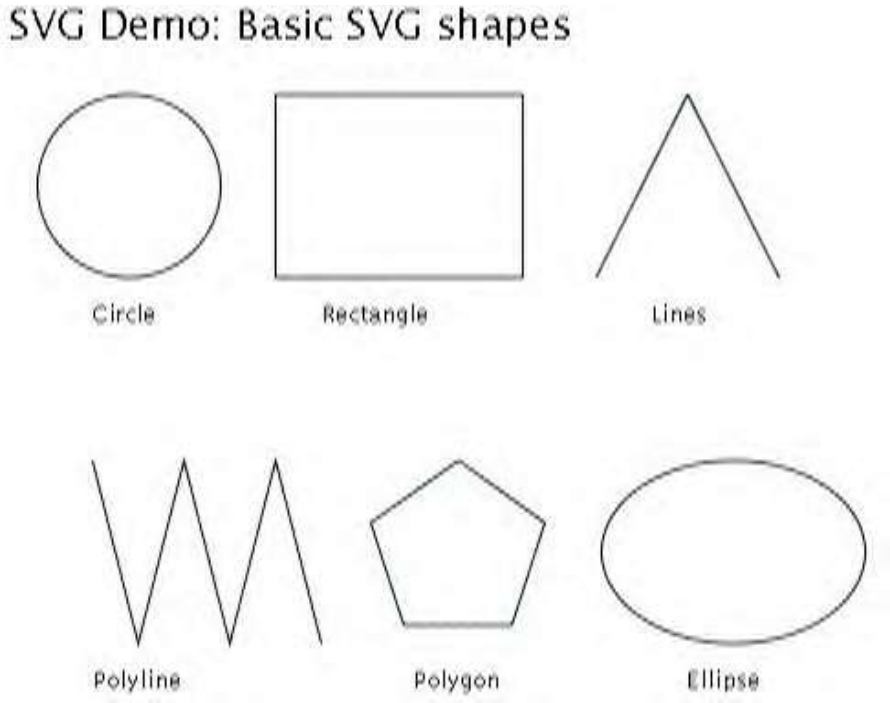


Figure 19: Dessin SVG

30.2.3Pourquoi SVG?

SVG permet de faire aisément toute sorte de dessin. Il est fait uniquement pour cela et est donc parfaitement adapté à notre besoin qui est de créer des diagrammes de séquence. Il est soutenu par le W3C et s'appuie sur XML. Un simple navigateur suffit pour afficher du SVG à condition d'avoir les *plugs-in* nécessaires. Il existe des logiciels permettant de visualiser ou de manipuler du SVG comme Adobe Viewer [8] ou Batik[9].

30.3Choix de Batik

Batik est un projet Apache[10] offrant des outils pour des applications SVG. Il offre aux développeurs plusieurs modules pouvant être utilisés ensemble ou séparément. Il propose par exemple un module pour l'analyse syntaxique de document SVG, un autre pour la génération de fichier SVG en Java, d'un outils de visualisation de fichier SVG…

Le choix de Batik nous a semblé opportun puisqu'il est simple d'emploi et possède toutes les fonctionnalités nécessaires pour manipuler et générer des fichiers SVG. D'autant plus qu'il est gratuit puisque sous licence Apache. Cependant, cet avantage est nuancé en raison de l'incompatibilité entre les licences Apache et les licences LGPL[11]. Il ne sera donc pas intégrer directement au logiciel CorbaTrace mais séparé de celui ci.

Nous utilisons Batik notamment pour générer du SVG à partir du code Java . La partie Test et Intégration de ce dossier montre des exemples de génération de fichiers SVG. Nous allons expliquer ici comment générer un fichier SVG à l'aide d'un exemple.

30.3.1La génération de fichiers SVG

Nous allons utiliser ici le *SVGDOMImplementation* de Batik qui n'est autre que l'implémentation de l'API DOM qui contient les interfaces modélisant en mémoire la représentation d'un document XML.

```
DOMImplementation impl = SVGDOMImplementation.getDOMImplementation();
```

Ensuite, il faut créer une instance de document depuis cette implémentation.

```
String svgNS = SVGDOMImplementation.SVG_NAMESPACE_URI;
Document doc = impl.createDocument(svgNS, "svg", null);
```

L'étape suivante consiste à créer un élément racine du document.

```
Element svgRoot = doc.getDocumentElement();
```

L'élément racine est pour nous la fenêtre où va s'afficher le dessin. Nous lui définissons en attribut une largeur et une hauteur.

```
svgRoot.setAttributeNS(null, "width", "200");
svgRoot.setAttributeNS(null, "height", "200");
```

Nous allons créer un rectangle avec un cercle à l'intérieur.

```
// create a rectangle
Element rectangle = doc.createElementNS(svgNS, "rect");
rectangle.setAttributeNS(null, "x", "40");
rectangle.setAttributeNS(null, "y", "50");
rectangle.setAttributeNS(null, "width", "100");
rectangle.setAttributeNS(null, "height", "100");
rectangle.setAttributeNS(null, "style", "fill:blue");

// create a circle
Element circle = doc.createElementNS(svgNS, "circle");
circle.setAttributeNS(null, "r", "20");
circle.setAttributeNS(null, "cx", "90");
circle.setAttributeNS(null, "cy", "100");
circle.setAttributeNS(null, "style", "fill:red;");
```

Lorsque l'on exécute le code du rectangle on obtient dans le fichier SVG la ligne suivante :

```
<rect x="40" y="50" width="100" height="100" style="fill:blue;"/>
```

Pour ajouter le rectangle à la fenêtre (l'élément principal) il suffit d'écrire :

```
svgRoot.appendChild(rectangle);
```

30.3.2Le contenu du fichier SVG

```
<svg contentScriptType="text/ecmascript" width="200" zoomAndPan="magnify"
contentStyleType="text/css" height="200" preserveAspectRatio="xMidYMid meet"
xmlns="http://www.w3.org/2000/svg" version="1.0">
      <rect width="100" x="40" height="100" y="50" style="fill:blue"/>
      <circle r="20" style="fill:red;" cx="90" cy="100"/>
</svg>
```

30.3.3Le résultat obtenu

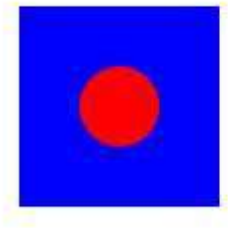


Figure 20: résultat du fichier

30.4 Description de l'API SequenceDiagram

L'API a été codée en java et permet de générer simplement des diagrammes de séquence en SVG. Elle est divisée en deux paquetages. Le premier dispose des objets liés au diagramme de séquence en général (acteur, action, messages…) et facilite ainsi leur construction. Le second est conçu particulièrement pour CorbaTrace même s'il peut très bien être utilisé dans un autre contexte.

30.4.1Le paquetage générique

Il offre un certain nombre d'objets utilisés dans les diagrammes de séquences comme le montre la figure 22.

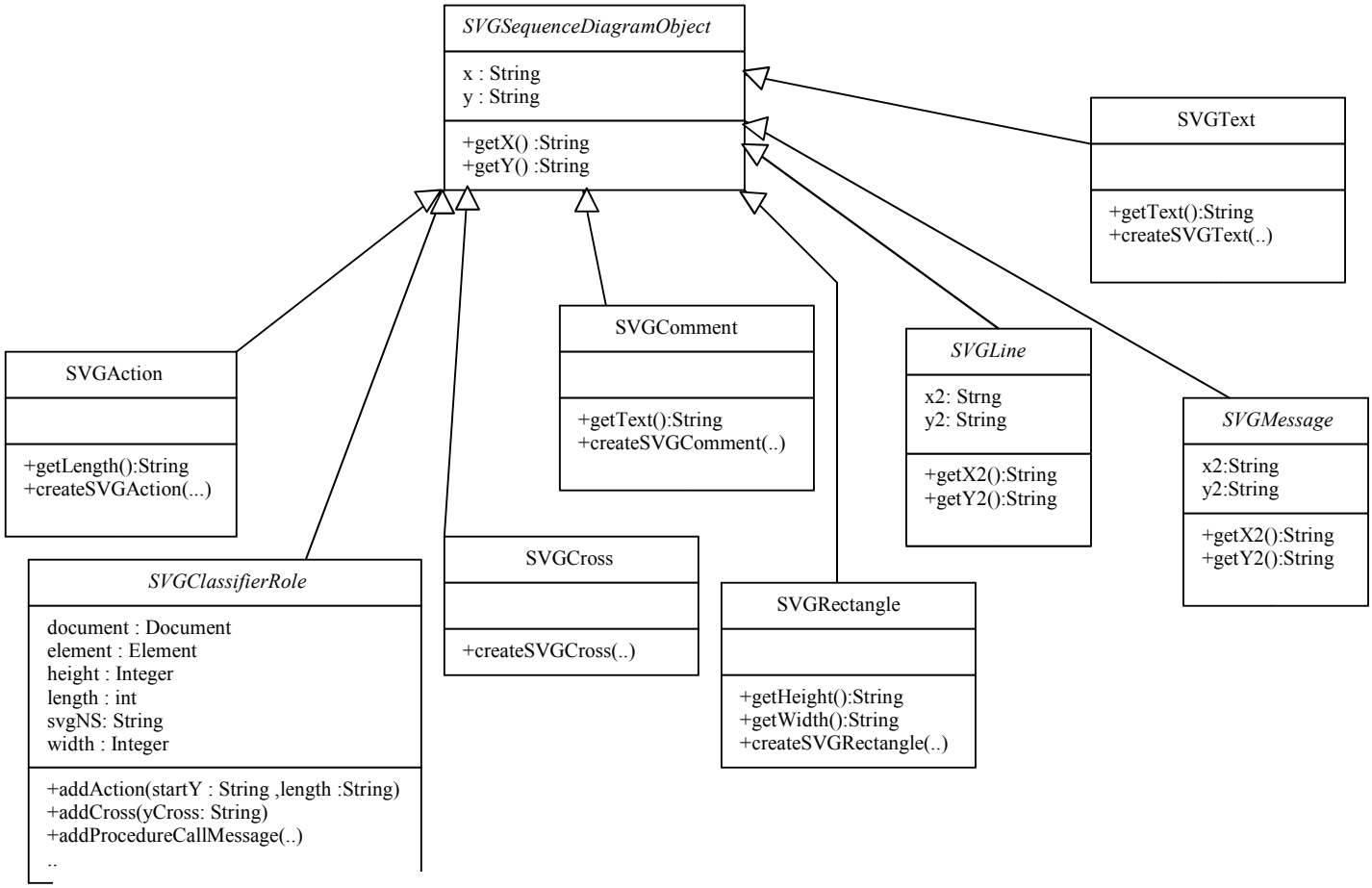


Figure 21:Diagramme de classes de l'API

La super classe du paquetage est la classe *SVGSequenceDiagramObject*. Elle est abstraite et contient en attribut les coordonnées de départ de l'objet. Les autres classes représentent des figures du diagramme de séquence. Elles possèdent toutes la méthode *createSVG[Nom_de_la _classe]* regroupant le code java générant l'objet SVG. Cette méthode prend en paramètre l'espace de nom auquel elle appartient, le document et son élément père dans l'arbre. La figure 23 détail les rôles des objets (acteur, instance…). La figure 24 détail quant à elle les messages (appel de méthode, asynchrone…).

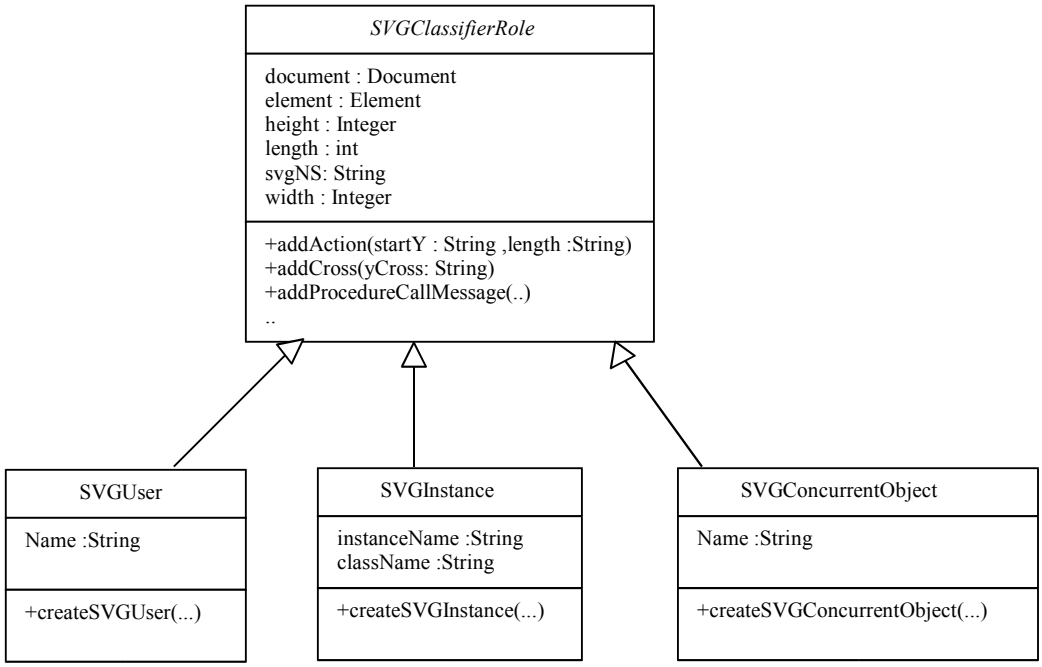


Figure 22: Diagramme de classes des rôles

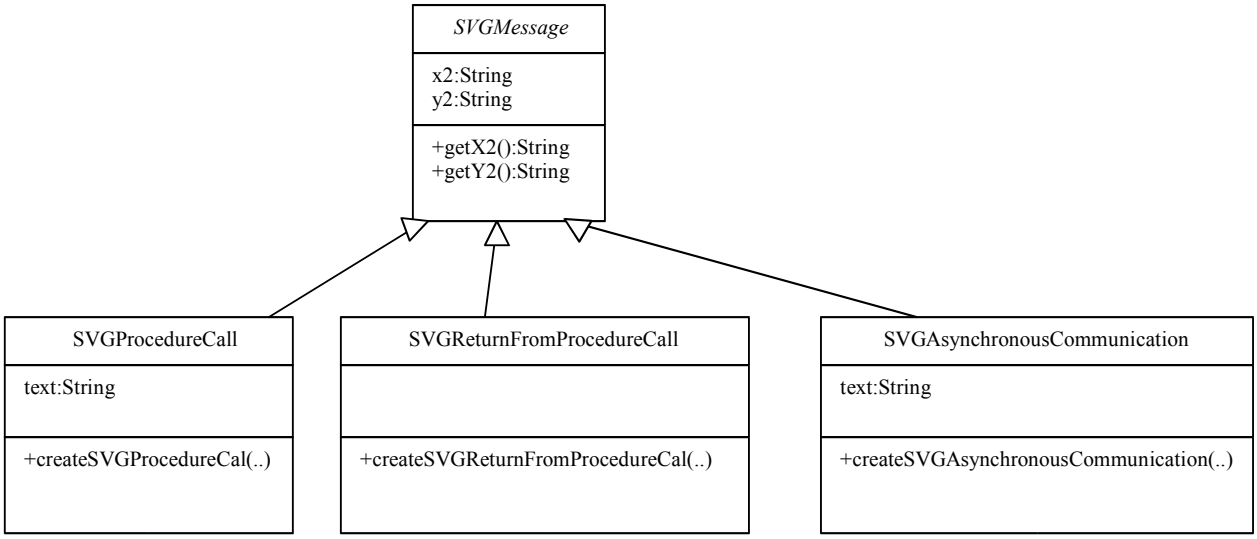


Figure 23:Diagramme de classes des messages

La classe *SVGSequenceDiagramDocument* du paquetage offre toutes les fonctionnalités pour créer un diagramme de séquence en utilisant ces classes. Elle dispose par exemple de la méthode *addInstance* qui va ajouter un rôle instance sur le diagramme de séquence. Il est possible ensuite d'utiliser les méthodes de la classe Instance pour ajouter des actions, des messages etc. à l'instance. Un exemple d'utilisation de ce paquetage est donné dans la partie 8 du dossier. La javadoc de l'API explique toutes les fonctionnalités du paquetage.

30.4.2Le paquetage spécifique à CorbaTrace

Ce paquetage utilise le précédent et ajoute des fonctionnalités liées à CorbaTrace. Par exemple le traitement des entiers de type *long* renvoyés par les intercepteurs CorbaTrace (le temps exact de l'interception) donnant des axes de temps beaucoup trop long. Ou encore, le commencement de l'axe de temps à partir du premier message envoyé…

Ce paquetage contient une classe *SVGCorbaTraceDocument* qui est la classe équivalente au *SVGSequenceDiagramDocument* mais pour les applications CorbaTrace. Elle contient les méthodes nécessaires à la création d'un fichier SVG pour CorbaTrace. Les classes `CorbaTraceAction`, `CorbaTraceObject`, `CorbaTraceMessage` et `CorbaTraceComment` stockent respectivement les actions, les objets, les messages et les commentaires. Lorsque l'on utilise les méthodes de la classe `SVGSequenceDiagramDocument` du précédent paquetage pour ajouter un élément dans le diagramme de séquence, on écrit dans le fichier SVG à la volée. Alors que pour ce paquetage, on stocke les éléments pour pouvoir les agencer correctement sur le diagramme en modifiant notamment les coordonnées des flèches symbolisant les messages. Ceci permet de gérer les problèmes énoncés dans le premier paragraphe. La méthode `createSVGFrame` écrit toutes les informations stockées dans le fichier SVG après avoir recalculé chaque position.

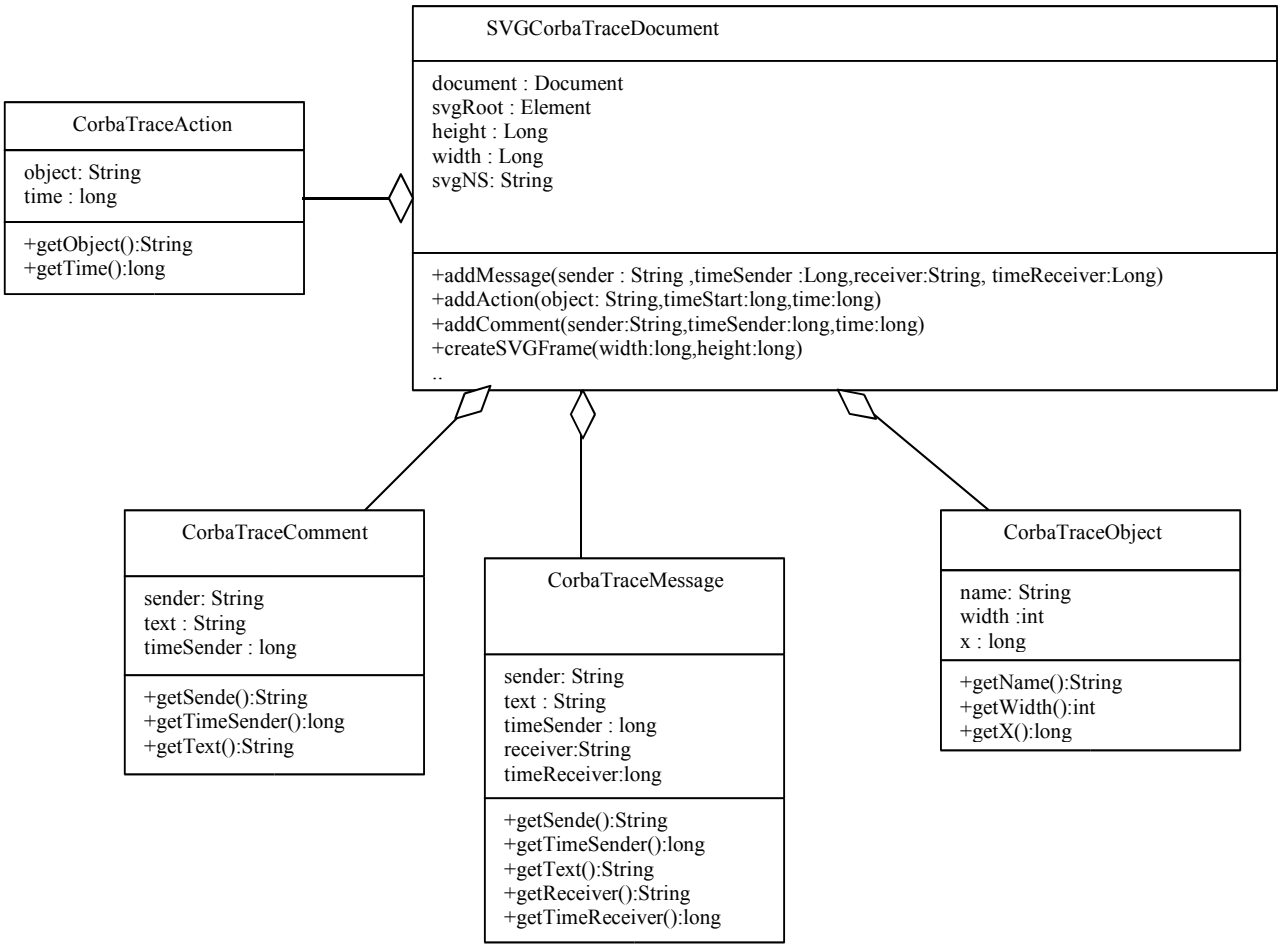


Figure 24:Diagramme de classes du second paquetage

31.1Intégration

31.1.1Ant

Ant est à Java ce que make est au C/C++, c'est un outil fourni par apache … qui permet de gérer la compilation, les paramètres de compilation, la mise à jour des CLASSPATH, la distribution, la génération de la documentation de projets Java (mais pas uniquement) de façon simple.

Ant étant openSource, il a été choisi pour simplifier la gestion du projet qui d'année en année prend de l'ampleur et qui devient donc difficile à maintenir. Cependant son usage n'est pas indispensable.

31.1.2Installation

31.1.2.1 Contenu de CorbaTrace v2.0

Le contenu de l'archive CorbaTrace v2.0 est le suivant :

`./`
Racine du projet, contient le fichier de configuration de Ant build.xml

`./etc/`
Fichiers de configurations, DTD, feuilles de styles XSLT

`./lib/`
Librairies utilisées par CorbaTrace

`./src/idl/`
Idl utilisées dans le projet

`./src/java/`
Sources java

`./src/java/corbaTrace/`
sources java du package corbaTrace

`./src/java/svgSequenceDiagram/`
sources java du package svgSequenceDiagram

31.1.2.2 Fichier de configuration corbaTrace.properties

Ant utilise un fichier de configuration nommé corbaTrace.properties situé dans le répertoire ./etc/ dans lequel sont regroupés tous les paramètres de compilation et de distribution qui varient d'un environnement de travail à l'autre. Pour débuter l'installation, ce fichier doit être mis à jour de la manière suivante :

- Choix de l'ORB (dé-commenter la ligne appropriée) :

```
# Sun/JDK 1.4 ORB
#orb=sun

# Orbacus ORB
orb=orbacus

# VisiBroker ORB
#orb=visibroker

# Orbit (the Gnome ORB)
#orb=orbit

# OpenORB (free ORB)
#orb=openorb
```

```
#...
```

- Emplacement des Libraires (mettre à jour les chemins) :

```
# librairie Batik.
lib.batik=lib/batik-1.1.1/batik.jar

# librarie FTP
lib.ftp=lib/ftp.jar
```

- Emplacement des Idl de l'ORB (mettre à jour les chemins) :

```
# ... in Orbacus context
idl.dir=/usr/local/ORBacus/JOB-4.1.2/idl/OB/

# ... in JDK 1.4 context
#idl.dir=${java.home}/idl

# ... in Visibroker context
#idl.dir=

# ... in Orbit context
#idl.dir=

# ... in OpenORB context
#idl.dir=
```

31.1.3Compilation

Une fois le fichier `corbaTrace.properties` correctement configuré il ne reste plus qu'a utiliser Ant en tapant les commandes suivantes :

`ant idl2java`
Compilation des Idls

`ant compile_svgSequenceDiagram`
Compilation du package svgSequenceDiagram

`ant compile`
Compilation du projet Corbatrace (package svgSequenceDiagram et Idls inclus)

`ant dist`
Création de la distribution incluant les packages corbaTrace.jar et svgSequenceDiagram.jar

`ant runServer`
Lancement du serveur de l'exemple d'application helloWorld

`ant runClient`
Lancement du client de l'exemple d'application helloWorld

`ant gui`
Lancement de log2SvgsequenceDiagram en mode graphique

`ant log2SvgsequenceDiagram`
Lancement de log2SvgsequenceDiagram en mode console

`ant doc`
Génération de la documentation du projet

31.1.4Diffusion

Afin de toucher un public aussi large que possible, un site Internet a été crée par l'équipe en charge de CorbaTrace l'année dernière. Ce site est accessible par l'URL : http://corbatrace.tuxfamily.org.

CorbaTrace est y distribué sous licence LGPL . On y retrouve également une présentation du logiciel, des news, et la documentation du logiciel.

Nous avons mis à jour le site concernant les modifications que nous avons apportés au logiciel.

31.2Tests

31.2.1Diagrammes de Séquence en SVG

31.2.1.1Le paquetage générique

Voici un exemple d'utilisation du paquetage permettant de créer des diagrammes de séquence.

```
import org.w3c.dom.*;
import org.w3c.dom.svg.*;
import org.apache.batik.dom.util.DOMUtilities;
import org.apache.batik.dom.svg.SVGDOMImplementation;
import org.apache.batik.svggen.SVGGraphics2D;
import java.io.*;
import corbaTrace.log2svg.svgSequenceDiagramObjects.*;



    public class SVGTest3{

      private Document doc;
      private String filename;
      private String svgNS;

      public static void main(String[] args){

          SVGTest3 st1 = new SVGTest3(args[0]);
          st1.create();
          st1.save();
      }



      public SVGTest3(String file){
           filename = file;

          //etape1: création d'une instance du SVG Document Object Model(disant
comment va être organisé le document XML)
          DOMImplementation impl = SVGDOMImplementation.getDOMImplementation();
          //etape2: creation du document XML depuis le SVGDOM
          svgNS = SVGDOMImplementation.SVG_NAMESPACE_URI;
          doc = impl.createDocument(svgNS,"svg",null);

      }

      public void create(){

          //etape3: creation du document SVG

          Element svgRoot = doc.getDocumentElement();
          SVGSequenceDiagramDocument svgDoc = new SVGSequenceDiagramDocument
(svgNS,doc,svgRoot);
          //création d'une fenêtre de départ de 700 par 500
```

```
          svgDoc.createSVGFrame("700","500");

          //création d'une insance
          SVGInstance instance1 = svgDoc.addInstance("Inst1","Classe1");
          //ajout de l'axe de temps de durée 300
          instance1.addTimeAxis("300");
          //ajout d'une action au temps 100 et de durée 30
          instance1.addAction("100","30");
          //ajout d'une autre action au temps 200 et de durée 40
          instance1.addAction("200","40");

          //création d'un acteur Jean créer par instance1
          SVGUser user = svgDoc.createUser("",instance1,"50","Jean");

          //création d'un acteur Paul
          SVGUser user1 = svgDoc.addUser("Paul");
          //ajout de l'axe de temps
          user1.addTimeAxis("250");
          //ajout d'une action au temps 110 et de durée 20
          user1.addAction("110","20");
          //ajout d'une action au temps 150 et de durée 100
          user1.addAction("150","100");
          //ajout d'une croix
          user1.addCross("250");
          //ajout d'un message simple entre user1 et instance1 envoyé au temps
150 et reçu au temps 200 disant hello
          user1.addAsynchronousMessage(instance1,"150","200","hello");

          //création d'un instance créer par le message new de user1
          SVGInstance instance3 = svgDoc.createInstance
("new",user1,"200","Inst3","Classe2");
          //ajout d'un axe de temps de durée 100
          instance3.addTimeAxis("100");
          //ajout d'un message simple envoyé par instance1 au temps 110 et reçu
par user1 au temps 110
          instance1.addAsynchronousMessage(user1,"110","110","salut");

          //création de l'objet concurrent créer par instance3
          SVGConcurrentObject svgCO1 = svgDoc.createConcurrentObject
("create",instance3,"50","objetC");

          //ajout d'un axe de temps
          svgCO1.addTimeAxis("50");

          //création d'un autre objet concurrent
          SVGConcurrentObject svgCO2 = svgDoc.addConcurrentObject("objet
concurrent");
          svgCO2.addTimeAxis("35");
          svgDoc.endDocument();
     }

     public void save(){

          try{
            //sauvegarde dans un fichier
            PrintWriter writer = new PrintWriter(new FileOutputStream
(filename));
            DOMUtilities.writeDocument(doc,writer);
            writer.flush();
            writer.close();

            System.exit(1);

          }
          catch(IOException exp){
```

```
            exp.printStackTrace();
          }
      }
}
```

La figure 28 montre le résultat obtenu.

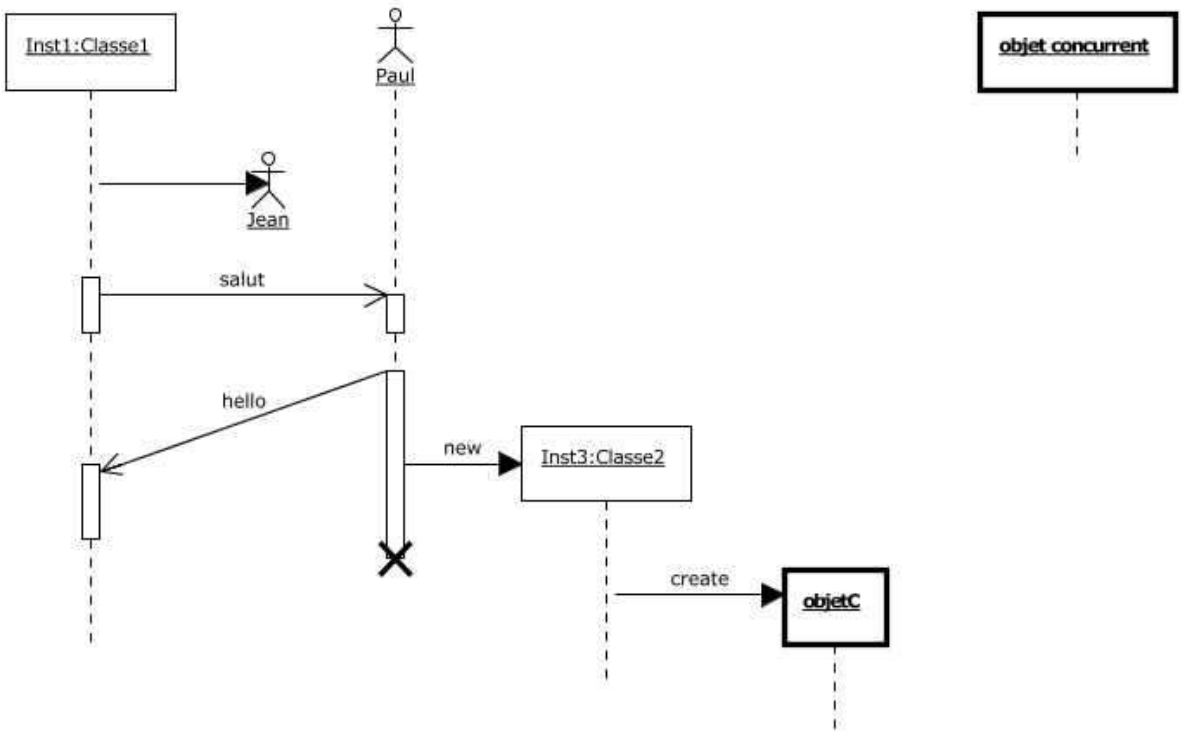


Figure 25: Diagramme de séquence SVG

31.2.1.2Le paquetage spécifique CorbaTrace

L'exemple suivant montre comment utiliser le second paquetage:

```
import org.w3c.dom.*;
import org.w3c.dom.svg.*;
import org.apache.batik.dom.util.DOMUtilities;
import org.apache.batik.dom.svg.SVGDOMImplementation;
import org.apache.batik.svggen.SVGGraphics2D;
import java.io.*;
import corbaTrace.log2svg.*;



    public class TestSVG{

      private Document doc;
      private String filename;
      private String svgNS;

      public static void main(String[] args){

          TestSVG st1 = new TestSVG(args[0]);
          st1.create();
          st1.save();
      }
```

```
      public TestSVG(String file){

          filename = file;

           //etape1: création d'une instance du SVG Document Object Model(qui
dit comment va être organise le document XML)
          DOMImplementation impl = SVGDOMImplementation.getDOMImplementation();
          //etape2: creation du document XML depuis le SVGDOM
          svgNS = SVGDOMImplementation.SVG_NAMESPACE_URI;
          doc = impl.createDocument(svgNS,"svg",null);

      }

      public void create(){

          //etape3: creation du document SVG
          // get the root element (the svg element)
          Element svgRoot = doc.getDocumentElement();
          //création d'une instance de SVGCorbaTraceDocument
          SVGCorbaTraceDocument svgDoc = new SVGCorbaTraceDocument
(svgNS,doc,svgRoot);
          //ajout d'un message envoyé par sender au temps 10 et reçu par
receiver au temps 20 appelant la méthode bonjour
          svgDoc.addMessage("sender",10,"receiver",20,"bonjour");
           //ajout du message de retour d'appel de méthode envoyé par receiver
au temps 30 et reçu par sender au temps 20
          svgDoc.addMessage("receiver",30,"sender",40);
          //ajout d'un message envoyé par sender2 au temps 110 qui n'abouti pas
          svgDoc.addMessage("sender2",110,null,140,"hello");
          //ajout d'un commentaire temps :10 pour le sender au temps 10
          svgDoc.addComment("sender",10,"temps:10");
          //ajout d'un commentaire temps :30 pour le receiver au temps 30
          svgDoc.addComment("receiver",30,"temps:30");
          //ajout d'un commentaire temps :110 pour le sender2 au temps 110
          svgDoc.addComment("sender2",110,"temps:110");
          //ajout d'une action pour sender2 au temps 50 d'une durée de 40
          svgDoc.addAction("sender2",50,40);
          //génération de la fenêtre
          svgDoc.createSVGFrame();
      }

      public void save(){

          try{

            PrintWriter   writer   =   new   PrintWriter(new   FileOutputStream
(filename));
            DOMUtilities.writeDocument(doc,writer);
            writer.flush();
            writer.close();

            System.exit(1);

          }
          catch(IOException exp){

            exp.printStackTrace();
          }
      }}
```

Cet exemple génère le diagramme de la figure 29.

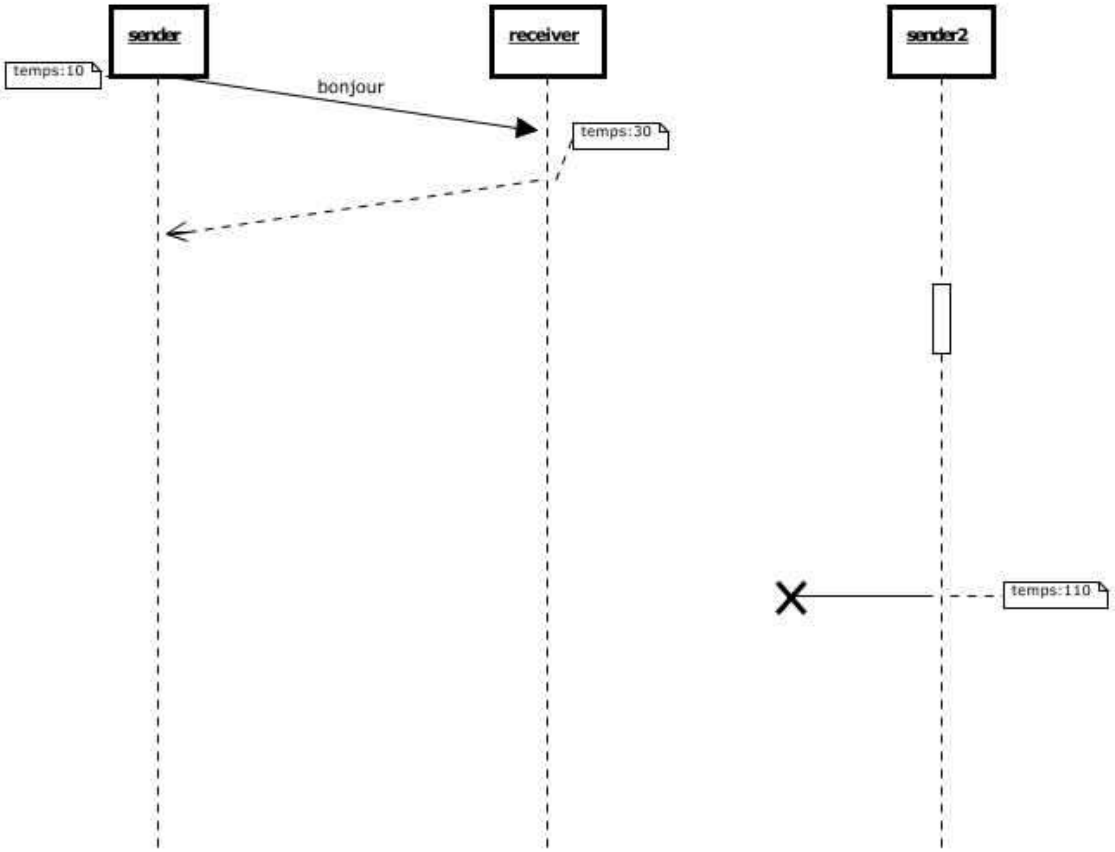


Figure 26:Le diagramme de séquence

32 Conclusion

Le travail d'équipe sur le projet CorbaTrace a été extrêmement enrichissant pour nous. Du point de vue technique, nous nous sommes formés à la spécification CORBA de manière plus poussée que lors du cours de DESS. Les améliorations que nous avons apportées à CorbaTrace telles que la génération de diagrammes de séquence au format SVG ou l'ajout d'une interface graphique pour l'utilisation du logiciel nous ont également permis de découvrir ou d'approfondir nos connaissances dans les technologies Batik et Swing.

Ce projet nous a également donné un aperçu de l'étape délicate de l'intégration dans un projet de cette taille, et de la répartition des tâches pour le mener à bien. Du point de vue humain, ce fut pour la plupart d'entre nous une des premières expériences de travail en équipe sur un projet d'une telle envergure. Nous avons pu mesurer l'importance de la communication et de l'organisation dans une telle configuration. Nous avons commis certaines erreurs dans notre organisation du travail en sous estimant parfois le temps nécessaire pour certaines étapes du projet, notamment la phase d'intégration. Cette expérience nous servira certainement à ne pas les reproduire.

Du point de vue du logiciel, la majorité de nos objectifs ont été atteints. En utilisant Java Logging nous avons pu éliminer le problème de la synchronisation due aux lectures/écritures des logs, nous avons fait en sorte qu'il soit possible de tracer l'exécution d'une application non répartie en utilisant les logs locaux. Nous avons aussi réalisé une interface graphique permettant de traiter les fichiers de logs de manière très simple. Enfin nous avons défini une API permettant de construire des diagrammes de séquences. API qui a ensuite été utilisée pour pouvoir interpréter les fichiers de logs en visionnant le diagramme de séquence généré dans une fenêtre de l'interface. Cependant il reste encore des points qui peuvent être améliorés. D'abord la synchronisation n'est pas encore totalement satisfaisante, certaines configurations particulières pouvant générer des diagrammes incohérents. L'API de génération de SVG peut encore être étendue pour gérer le changement d'échelle des diagrammes de séquences par exemple ou encore la mise en évidence de certains messages particuliers via l'utilisation de couleurs, etc. Les outils de l'interface graphique comme le module FTP et la gestion de projets pourront êtres améliorés et d'autres fonctionnalités ajoutées comme la gestion des langues pour traduire les éléments de l'interface.

D'année en année le projet CorbaTrace prend de l'ampleur tout en gagnant en souplesse et performances. Aussi, nous espérons et ferons en sorte que cela continue.

33

34 Bibliographie

[1] Poseidon For UML, www.gentleware.com/

[2] Rational Rose, www.rational.com/

[3] MagicDraw, www.magicdraw.com/

[4] Projet LATEX, www.latex-project.org

[5] Spécifications SVG, www.w3.org/TR/SVG/

[6] OMG, www.omg.org

[7] World Wide Web Consortium, www.w3c.org

[8] Adobe viewer, www.adobe.com/svg/main.html

[9] Batik, xml.apache.org/batik/

[10] Apache, www.apache.org/

[11] Licence LGPL, www.opensource.org/licenses/lgpl-license.php

[12] The Swing Connection, http://java.sun.com/products/jfc/tsc/

[13] The Swing FAQ, http://www.mindspring.com/~scdrye/java/faq.html