Université de Montréal

Rétro-ingénierie des diagrammes de séquence par visualisation interactive

par

Hassen Grati

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Juillet, 2010

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé :

Rétro-ingénierie des diagrammes de séquence par visualisation interactive

présenté par:
Hassen Grati

a été évalué par un jury composé des personnes suivantes :

Max Mignotte
président-rapporteur

Houari A. Sahraoui
Directeur de recherche

Pierre Poulin
Codirecteur

Bruno Dufour
Membre du jury

Mémoire accepté le

RÉSUMÉ

Nous proposons une approche semi-automatique pour la rétro-ingénierie des diagrammes de séquence d'UML. Notre approche commence par un ensemble de traces d'exécution qui sont automatiquement alignées pour déterminer le comportement commun du système. Les diagrammes de séquence sont ensuite extraits avec l'aide d'une visualisation interactive, qui permet la navigation dans les traces d'exécution et la production des opérations d'extraction. Nous fournissons une illustration concrète de notre approche avec une étude de cas, et nous montrons en particulier que nos diagrammes de séquence générés sont plus significatifs et plus compacts que ceux qui sont obtenus par les méthodes automatisées.

Mots-clés : rétro-ingénierie, diagramme de séquence, trace d'exécution, visualisation interactive.

ABSTRACT

We propose a semi-automated approach for the reverse engineering of UML sequence diagrams. Our approach starts with a set of execution traces that are automatically aligned to determine the common behavior. Sequence diagrams are then extracted with the help of an interactive visualization, which allows navigating though execution traces and performing extraction operations. We provide a concrete illustration of our approach with a case study, and show in particular that the resulting diagrams are more meaningful and more compact than those extracted by automated approaches.

Keywords: reverse engineering, sequence diagram, execution trace, interactive visualization.

TABLES DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

Avec mes vifs remerciements, je tiens à adresser cette dédicace, ma gratitude et mon amour à :

- Mes chers parents en témoignage de ma reconnaissance pour les grands sacrifices consentis pour moi ainsi que leur encouragement, leur amour, leur soutien moral et matériel, et tous les efforts qu'ils ont consacrés pour mon éducation et mon bien-être.

- Mon frère et ma sœur, à qui je souhaite une vie pleine de réussites et de bonheur.

- Tous mes amis(es) et collègues qui m'ont soutenu et m'ont comblé par leurs conseils pour la réalisation de ce travail.

Et enfin à tous ceux qui me sont chers.

REMERCIEMENTS

Respectueusement, je remercie chaleureusement Houari Sahraoui, professeur titulaire à l'université de Montréal, de m'avoir tout d'abord enseigné les bases de mes connaissances dans le domaine du génie logiciel et de m'avoir ensuite dirigé dans ma recherche. Il a semé mon « mail » d'informations sans cesse renouvelées afin de faire progresser mon travail.

Je tiens à remercier professeur Pierre Poulin pour avoir dirigé ma recherche et qui n'a jamais épargné l'effort de m'aider, pour la pertinence de ses orientations ainsi que pour la grande disponibilité dont il a fait preuve tout au long du déroulement de ce mémoire. Qu'il trouve dans ces quelques lignes l'expression de mon profond respect et de ma réelle gratitude.

J'adresse pareillement mes sincères remerciements à professeur Max Mignotte, pour l'honneur qu'il nous a fait en présidant notre jury de mémoire de maîtrise.

Je tiens aussi à remercier vivement professeur Bruno Dufour d'avoir bien voulu apporter ses conseils son jugement sur ce modeste travail.

Je remercie les membres de l'équipe du laboratoire GÉODES pour leur sympathie et leur gentillesse : Merci à Marouane pour ses conseils, merci à Simon pour son humanité, merci à Martin et Guillaume pour leurs encouragements…

J'exprime également ma gratitude à tous mes enseignants, qui ont contribué chacun dans son domaine, à ma formation universitaire, sans laquelle je ne serais jamais arrivé à réaliser ce travail.

À tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Chapitre 1

Introduction

Dans ce chapitre, nous présentons dans un premier temps le contexte de notre étude en détaillant l'importance de la rétro-ingénierie, ainsi que l'importance des diagrammes UML, et en particulier des diagrammes de séquence des systèmes logiciels. Ensuite, nous discutons les principaux problèmes de la rétro-ingénierie des modèles UML en présentant les solutions proposées dans notre étude. Enfin, nous donnons un aperçu des différents chapitres de notre mémoire.

1.1. Contexte

Les logiciels deviennent de plus en plus complexes et leur maintenance est de plus en plus coûteuse. De ce fait, il est vraiment important de pouvoir comprendre le fonctionnement des systèmes logiciels. Malheureusement, la plupart des systèmes logiciels manquent de documentation et de manuels d'utilisation, ce qui rend encore plus complexe la tâche de gestion de ces systèmes. Selon certaines statistiques, jusqu'à 60% des travaux de maintenance et d'entretien sont consacrés à la compréhension des logiciels [21]. De ce fait, il est primordial de concevoir et de développer des outils et des techniques qui faciliteront la tâche de compréhension de tels systèmes.

Une des techniques reconnues efficaces pour comprendre un programme est la rétro-ingénierie, dont un aperçu de paradigme sera détaillé dans la prochaine sous-section.

1.1.1. Rétro-ingénierie

La rétro-ingénierie ou l'ingénierie inverse est le processus qui consiste à découvrir les principes technologiques d'un système grâce à l'analyse de sa structure et de son fonctionnement. En d'autres termes, la rétro-ingénierie est une technique permettant de comprendre le fonctionnement d'un programme dont nous essayons de déterminer les processus produisant un certain résultat à partir d'une certaine entrée.

De nos jours, la rétro-ingénierie des systèmes informatiques est devenue l'un des principaux domaines d'études du génie logiciel. En effet la rétro-ingénierie permet : (1) la documentation et la compréhension de la structure et du fonctionnement d'un logiciel insuffisamment documenté [2][3][4], (2) l'identification des défauts de conception [26][27] ainsi que des failles de sécurité d'un programme [28], et (3) la récupération du code perdu lorsque seul l'exécutable est disponible [28] (par exemple, la récupération des fichiers .java à partir des fichiers .class). Un exemple de problème de rétro-ingénierie est la génération du diagramme de séquence à partir du code. Nous décrirons brièvement le diagramme de séquence dans la prochaine sous-section.

1.1.2. Diagramme de séquence

Le diagramme de séquence est l'un des plus importants diagrammes UML pour la compréhension des systèmes informatiques. Il offre une vue abstraite d'un niveau élevé du comportement du système au cours d'un scénario précis, par la présentation graphique des objets ainsi que les messages échangés entre eux. En effet, les diagrammes de séquence permettent de représenter des collaborations entre objets selon un point de vue temporel, en mettant l'accent sur la chronologie des envois de messages. Les diagrammes de séquence peuvent servir aussi à illustrer des cas d'utilisation. Finalement, ils offrent également un excellent moyen pour communiquer les aspects dynamiques d'un système. La Figure 1.1 donne un exemple du diagramme de séquence qui décrit le scénario « emprunt » d'une œuvre dans une bibliothèque. Lors que l'acteur principal « :Adhérent » demande l'emprunt d'une œuvre, l'objet « :Bibliothèque » vérifie sa disponibilité et puis initialise le prêt. Enfin, une mise à jour du nombre d'exemplaires s'effectue par l'objet « :Prêt ».

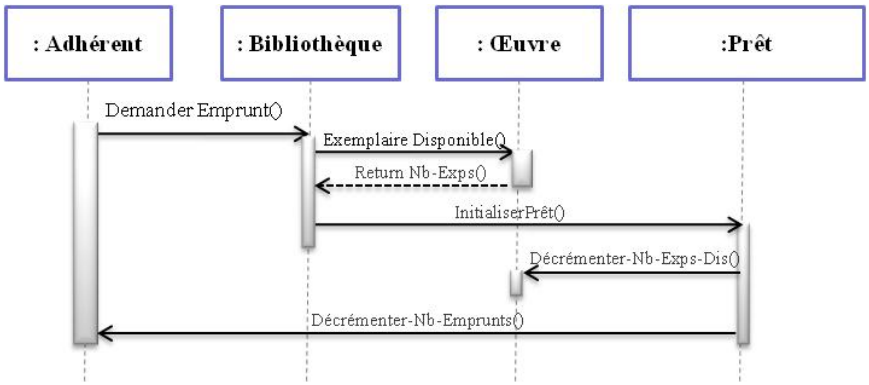


Figure 1.1 : Exemple d'un diagramme de séquence du scénario « emprunt » d'une œuvre dans une bibliothèque.

Plusieurs techniques permettent la génération du diagramme de séquence à partir du code. Par contre, elles engendrent des limites. Nous énonçons succinctement les motivations liées à notre travail dans ce qui suit.

1.2. Motivation

1.2.1. Rétro-ingénierie des diagrammes de séquence

Lors de son cycle de vie, un système logiciel subit plusieurs métamorphoses qui se rapportent surtout à ses fonctionnalités de base. Ceci engendre en général des changements importants par rapport à la structure initiale de son code, ce qui rend son analyse plus complexe. Afin de pouvoir faciliter la tâche de compréhension des systèmes logiciels, on a souvent recours à la rétro-ingénierie.

La rétro-ingénierie des modèles d'analyse et de conception contribue à l'amélioration de nombreuses activités de développement et de maintenance de logiciels. Dans ce contexte, de nombreuses contributions ont été proposées pour la rétro-ingénierie de la structure statique des systèmes orientés objet (OO) [3]. À l'exception de quelques problèmes, tels que la récupération des relations, ces contributions sont assez matures pour être intégrées dans des outils commerciaux tels que *Rational*[1], *Together*[2], et *NetBeans*[3].

[1] http://www-01.ibm.com/software/rational/

[2] http://www.borland.com/us/products/together/

Cette réalité contraste avec la difficulté d'extraire des modèles de comportement des systèmes OO, tels que les diagrammes de séquence. En effet, les programmes modernes utilisent largement les fonctionnalités du langage dynamique (le polymorphisme, le chargement et la génération dynamique de classes et la réflexion) qui rendent difficile et souvent impossible de saisir le comportement par analyse statique [25]. Pour contourner cette limitation, de nombreuses équipes de recherche ont adopté une approche alternative basée sur l'analyse dynamique [3][4][5][6]. Dans ces approches, les traces d'exécution sont utilisées pour retrouver les éléments des diagrammes de séquence.

Bien que ces approches aient permis d'améliorer la qualité de l'extraction des modèles de comportement, deux problèmes demeurent. D'abord, les modèles extraits sont définis en terme d'implémentation et ne fournissent pas une vue abstraite. En d'autres termes, le processus d'extraction ne peut pas distinguer entre les informations abstraites (telles que les messages de communication entre les objets d'affaires) et les détails d'implémentation. Le second problème de ces approches est que dans plusieurs cas, les diagrammes extraits correspondent à des exécutions uniques, ce qui limite leur généralisation.

1.2.2. Problèmes des approches automatiques de la rétro-ingénierie

De nos jours, la plupart des approches de rétro-ingénierie se font de façon automatique. L'automatisation se base en général sur des approximations qui peuvent produire des résultats plus ou moins précis. En effet, il est reconnu que la plupart des approches automatiques présentent un degré de complexité assez élevé, ce qui rend difficile d'appréhender toutes les manières de les utiliser. De plus, comme c'est le cas de la plupart des approches automatiques, les résultats générés sont volumineux, tel le cas de traces d'exécution qui sont souvent illisibles.

Divers travaux basés sur de telles approches automatiques ont fait ressortir l'existence de plusieurs informations inutiles qui ne servent généralement pas dans le contexte approprié [1][2][3][4]. En effet, les diagrammes de séquence générés automatiquement

[3] http://netbeans.org/

présentent des objets temporaires, des objets d'interfaces graphiques non essentiels pour la compréhension de l'application et certains messages inutiles. Un autre problème réside dans le fait que les objets identifiés chacun par une adresse mémoire unique rendent difficile à un utilisateur de distinguer entre deux ou plusieurs objets, surtout dans le cas où ils sont instanciés à partir de la même classe (type). Ceci rend particulièrement difficile la gestion de ces objets, et par conséquent, les analyses demeurent complexes.

L'objectif de notre étude est de construire les diagrammes de séquence d'une manière pertinente, non seulement pour conserver la sémantique des données, mais aussi pour contourner les problèmes précédemment mentionnés.

1.3. Contribution

Dans ce mémoire, nous proposons une approche qui peut contourner les limitations inhérentes à l'automatisation. Au lieu de reposer sur l'automatisation complète du processus pour la rétro-ingénierie des diagrammes de séquence, nous proposons un processus semi-automatique, avec la participation d'un analyste intervenant dans un environnement de visualisation interactif. Notre système exécute certaines des actions automatiques lorsqu'elles sont matures et fiables. Il utilise aussi certaines heuristiques pour recommander des actions à l'analyste, qui peut alors les accepter ou les refuser. L'analyste peut aussi intervenir via des actions manuelles tout au long du processus, en fonction de ses connaissances contextuelles.

Pour capturer l'aspect dynamique d'un système logiciel, notre approche combine des traces d'exécution obtenues par une instrumentation du code. Pour ce faire, nous avons développé un algorithme d'alignement local, basé sur des chaînes de caractères.

Afin d'évaluer notre approche, nous avons réalisé une étude sur un cas décrit par Briand et al. [3]. En particulier, nous comparons les diagrammes de séquence obtenus en utilisant notre approche avec ceux dérivés automatiquement en analysant les différences. Nos résultats montrent que, dans un temps raisonnable, un analyste peut extraire des diagrammes de séquence contenant l'information abstraite, sans l'encombrement des détails inutiles d'implémentation.

- Nous résumons les contributions de notre recherche comme suit : Nous avons développé une stratégie automatique pour la combinaison des traces d'exécution générées à partir d'un même cas d'utilisation, en employant un algorithme d'alignement de chaînes de caractères adapté afin d'obtenir une seule trace combinée qui englobe tous les scénarios.
- Pour permettre de voir la trace combinée, nous proposons une métaphore de visualisation en trois dimensions permettant une perception efficace, pratique et complète.
- Finalement, nous définissons un processus semi-automatique pour la rétro-ingénierie des diagrammes de séquence en nous basant sur les techniques de visualisation interactives.

En terminant, nous avons publié les résultats de nos travaux dans une conférence scientifique [24] où nous avons résumé la majeure partie des travaux présentés dans ce mémoire. Ceci inclut les enjeux des approches existantes pour la rétro-ingénierie des diagrammes de séquence, les stratégies et techniques développées dans notre approche, ainsi que l'analyse d'une étude de cas.

1.4. Structure du mémoire

Le reste de ce mémoire est organisé comme suit. Le chapitre 2 représente un survol des principaux travaux en relation avec notre sujet. Les détails de notre approche sont présentés au chapitre 3. Les résultats d'une étude de cas sont énoncés et discutés dans le chapitre 4. Enfin au chapitre 5 nous récapitulons les principales idées introduites au sein de notre mémoire et nous proposons quelques perspectives liées à notre travail.

Chapitre 2

État de l'art

2.1. Introduction

Nous présentons dans ce chapitre un survol des travaux existants concernant la rétro-ingénierie du code vers un modèle, et plus précisément vers les diagrammes de séquence. Notre approche touche principalement à deux domaines : la rétro-ingénierie et la visualisation des logiciels. Nous commençons par introduire les notions de base et par donner quelques définitions nécessaires pour la compréhension de notre approche. Ensuite, nous détaillons les travaux traitant le problème de transformation du code vers des diagrammes de séquence, et surtout ceux similaires à notre approche, basée sur l'utilisation des traces d'exécution. Enfin, nous survolons quelques travaux qui utilisent la visualisation comme support pour analyser les traces d'exécution.

2.2. Rétro-ingénierie des diagrammes de séquence

La rétro-ingénierie ou encore « l'ingénierie inverse » est une technique permettant de comprendre le fonctionnement d'une technologie en utilisant des méthodes de désassemblages successifs afin de mettre en évidence toutes les étapes de la fabrication d'un système, et ainsi de déterminer comment celui-ci a été conçu.

Cette technique est utilisée dans plusieurs domaines et plus particulièrement dans les technologies modernes comme la mécanique, l'électronique, l'informatique, etc. La rétro-ingénierie consiste à transformer un code vers un modèle pour augmenter le niveau d'abstraction afin de mieux comprendre et analyser un logiciel. En effet, la rétro-ingénierie permet de mieux analyser et comprendre un code inconnu et de détecter plus facilement les erreurs de fonctionnement du système.

Nous nous intéressons dans notre projet à la transformation/génération semi-automatique d'un diagramme de séquence à partir du code. Avant de donner plus de détails sur les travaux existants dans ce domaine, nous commençons par introduire quelques éléments de base nécessaires concernant les diagrammes de séquence.

2.2.1. Diagramme de séquence

Nous allons essayer au cours de cette section de détailler les caractéristiques des diagrammes de séquence afin de justifier leur importance et leur performance. Un **diagramme de séquence** est une représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans le cadre d'un scénario particulier, dans le but de montrer le déroulement des actions, sous la formulation du langage de modélisation **UML** *« Unified Modeling Language »*.

Un diagramme de séquence est une forme de diagramme comportemental qui permet de spécifier les interactions qui existent entre un groupe d'objets, c'est-à-dire, de représenter les collaborations entre les objets **d'un point de vue temporel**. Dans ce type de diagramme, on s'intéresse à **l'aspect chronologique** des envois de messages entre les objets. En effet, même si d'autres diagrammes comportementaux peuvent convenir, les diagrammes de séquence sont les plus utilisés parce qu'ils permettent de voir comment les objets se communiquent mutuellement. Grâce à ces informations, nous pouvons déterminer plus précisément pourquoi deux objets sont liés.

Comme les diagrammes de séquence sont toujours lus du haut vers le bas, ils illustrent l'ordre dans lequel les messages sont envoyés entre les objets.

Les primordiaux composants d'un diagramme de séquence [20] sont :

- **Les objets (classes) :** Sur un diagramme de séquence, les objets apparaissent toujours dans la partie supérieure, ce qui facilite l'identification des classes (en supposant que la notation des classes est utilisée) qui participent à l'interaction.
- **Les messages :** Ils représentent les flux d'informations échangées entre les objets du système. UML propose un certain nombre de stéréotypes graphiques pour décrire la nature d'un message, tels que les messages simples, les

messages minutés (*timeout*), les messages synchrones, les messages asynchrones, etc.

- **Les fragments :** Ils permettent de donner une vue compacte du diagramme de séquence. Un fragment est constitué d'un ensemble de messages reliés par un opérateur de contrôle. Il existe plusieurs types d'opérateurs de contrôle . Dans ce travail, nous nous intéressons aux alternatives/conditionnelles (*alt*, *opt*), aux boucles (*loop*).

Pour des raisons de simplification, l'acteur principal, personne ou objet en dehors du système, responsable du déclenchement du scénario, est présenté à gauche du diagramme et les acteurs secondaires éventuels à droite du système. Ainsi, le but est de décrire comment se déroulent les actions entre les acteurs (objets).

La Figure 1.1 représente un exemple d'un diagramme de séquence. L'acteur principal «:Adhérent » est présenté à gauche pour des raisons de simplification. Les objets sont généralement représentés par des rectangles. Une ligne verticale représente chaque ligne de vie d'un objet. Des flèches sont employées pour la représentation des échanges de messages.

L'importance des diagrammes de séquence réside dans le fait qu'ils présentent une interface visuelle simple pour la compréhension du fonctionnement du système, ce qui permet de trouver plus facilement les erreurs que de parcourir le code lui-même.

Dans la prochaine sous-section, nous présenterons une classification des travaux existants permettant l'extraction des diagrammes de séquence à partir du code.

2.2.2. Approches existantes

Nous distinguons deux grandes catégories dans les approches existantes : statique et dynamique. Les approches appartenant à ces deux catégories sont considérées comme automatiques et l'intervention de l'usager n'est pas considérée.

2.2.2.1. Approches statiques

L'analyse statique consiste à utiliser la structure du code pour générer le diagramme de séquence. L'un des principaux travaux basés sur l'analyse statique est celui de Rountev et al. [2]. Ils ont proposé une approche permettant l'extraction des diagrammes de séquence UML à partir du code en passant par la construction des **graphes de flot de contrôle** Dans cette étude les nœuds représentent les blocs de base d'un programme (il s'agit d'ensembles d'instructions qui ne contiennent pas d'appels de méthodes ni de branchements conditionnels) et les liens représentent toutes sortes d'interactions entre ces blocs.

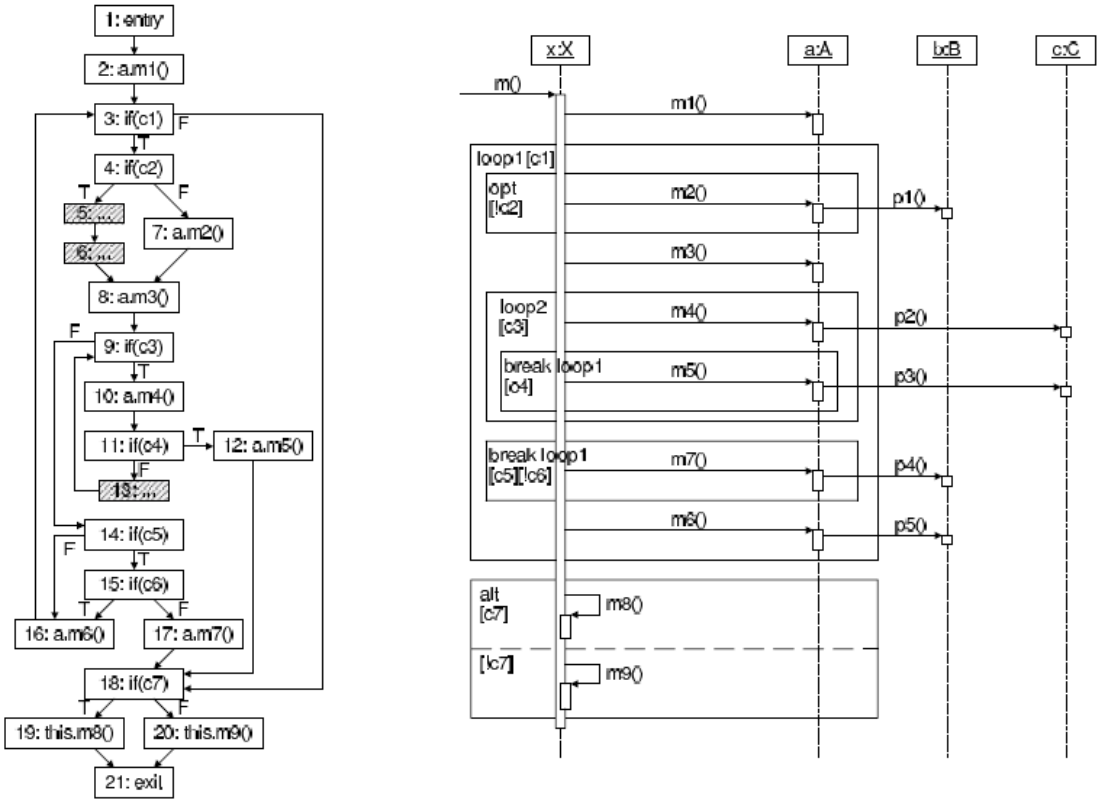


Figure 2.1 : Graphe de flot de contrôle et le diagramme de séquence généré Rountev et al. [2].

L'approche proposée par Rountev et al. [2] est composée de quatre phases. La construction du graphe du flot de contrôle représente la première phase. Dans la deuxième phase, ils décomposent le graphe en des sous-graphes en assurant la différenciation des différents fragments. On cite par exemple : fragment

alternatif « alt » où il existe deux comportements possibles, fragment option « opt » où il existe un seul comportement possible, fragment « loop » où il s'agit d'un ensemble d'interactions qui s'exécutent en boucle, etc.

Le diagramme de séquence est construit au cours de la troisième phase. La construction est basée principalement sur la règle suivante « Un message sera créé dans le diagramme de séquence lors de chaque appel d'une méthode ».

Enfin, une phase de finition vise à rendre le diagramme de séquence construit plus compréhensible et plus clair par l'application d'une série de transformations afin de réduire les imbrications entre les objets.

Le premier diagramme de séquence représenté dans la Figure 2.1 est le diagramme généré lors de l'application de cette approche, ainsi que le graphe de flot de contrôle correspondant.

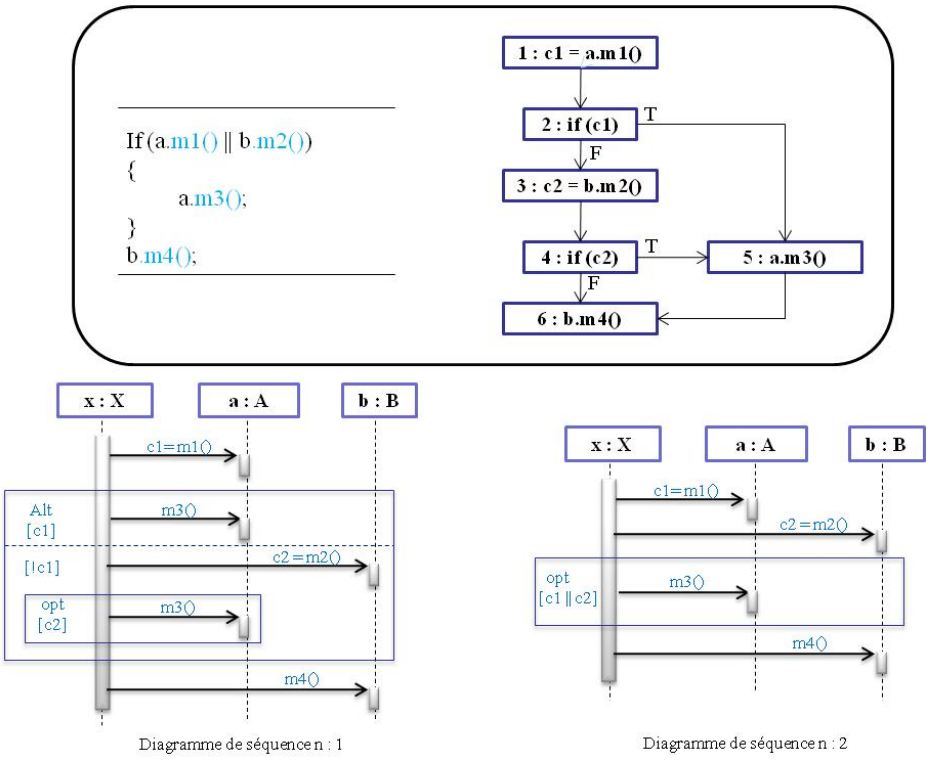


Figure 2.2 : Exemple du problème de réplication dans Rountev et al. [2].

L'une des principales limites de cette approche est la réplication de messages dans certains cas. Pour illustrer cette limite, nous utilisons l'exemple de la Figure 2.2. En effet, comme le montre la figure, lors de la construction du graphe du flot de contrôle, la condition « **a.m1() || b.m2()** » est représentée par plusieurs blocs, ce qui implique la réplication du message « **a.m3()** » dans le premier diagramme de séquence présenté ci-après.

Cette méthode est limitée par l'analyse de flots de contrôle parce que les informations qui ne peuvent pas apparaître dans le graphe de flots ne seront pas présentes dans le diagramme de séquence, par exemple les structures conditionnelles constituées de plusieurs clauses.

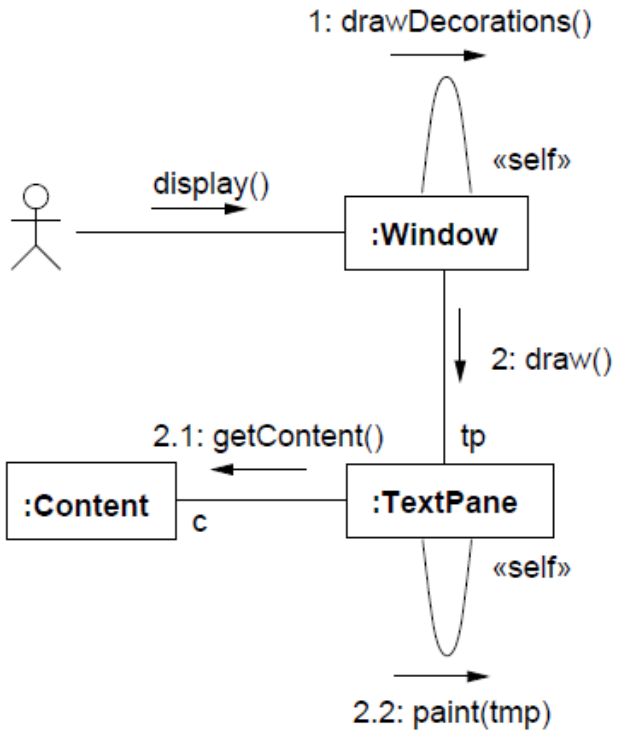


Figure 2.3 : Diagramme de collaboration généré de Kollman et al. [9].

Dans le même contexte, Kollman et al. [9] proposent une technique d'extraction automatique des diagrammes de collaboration qui sont conceptuellement similaires aux diagrammes de séquence basée sur une analyse statique du code. Ils proposent un méta-

modèle Java qui englobe toutes les informations sur les interactions entre les objets du programme, ainsi qu'un algorithme récursif basé sur des règles pour transformer une partie du programme en un diagramme de collaboration. La Figure 2.3 montre un exemple de diagramme de collaboration généré par Kollman et al. [9].

Tonella et al. [10] présentent une approche statique pour la rétro-ingénierie des diagrammes de séquence et des diagrammes de collaboration à partir de code C++. Les objets créés sont identifiés à partir des variables du programme et les messages échangés entre eux sont dérivés des appels de méthodes. Leur contribution réside dans le fait que leur algorithme de rétro-ingénierie peut s'appliquer aux systèmes incomplets (pas de nécessité que le programme compile). Par conséquent, il sera possible d'analyser et comprendre le comportement d'une partie d'un système en isolant quelques composants.

Une autre limite liée à l'utilisation des analyses statiques du code réside dans le manque de précision. Par exemple, même s'il est possible de résoudre certains appels polymorphiques, il est impossible de choisir un objet unique destinataire de l'appel comme le nécessitent les diagrammes de séquence. Pour contourner cette limitation, de nombreuses équipes de recherche utilisent l'analyse dynamique pour extraire les diagrammes de séquence par l'analyse des traces d'exécution.

2.2.2.2. Approches dynamiques : analyse des traces d'exécution

L'analyse dynamique consiste à analyser l'exécution de l'application. Plusieurs travaux essayent de générer le diagramme de séquence en analysant les traces d'exécution.

Briand et al. [3] se reposent sur un ensemble de règles de transformations formelles pour la rétro-ingénierie des diagrammes dynamiques d'UML. Ces règles couvrent tous les différents aspects de génération d'un diagramme de séquence, comme les branchements conditionnels, les boucles, etc. Leur idée est de définir des règles de transformations au niveau du méta-modèle de trace d'exécution et du méta-modèle de diagramme de séquence. D'une façon générale, les méta-modèles utilisés sont exprimés sous la forme de diagrammes de classe dont toutes les informations d'interactions sont bien exprimées.

La génération des traces d'exécution [3] nécessite l'instrumentation du code d'une façon automatique afin de permettre la récupération des informations suivantes pour chaque appel d'une méthode :

- La méthode appelante et la méthode appelée.
- L'objet appelé (l'adresse mémoire et le nom de la classe).
- Les structures conditionnelles.
- Les structures itératives ainsi que le type et la condition d'arrêt.

La Figure 2.4 illustre le méta-modèle de diagramme de séquence dont la classe abstraite **Message** présente toutes les sortes de messages. Chaque message dans le diagramme de séquence a comme source un *callerObject* et comme destination un *calleeObject.* Les messages peuvent être de trois types : (1) un appel de méthode « Class *MethodMessage* », (2) un message de retour « Class *ReturnMessage* », ou (3) un message itératif « Class *IterationMessage* ». Ces messages peuvent contenir des paramètres « Class *ParameterSD* » et sous certaines conditions « Class *ConditionClauseSD* ».).

Comme illustré à la Figure 2.5, le méta-modèle de trace est similaire au méta-modèle de diagramme de séquence. Tous les concepts sont en équivalence 1-à-1 à l'exception notable que *MethodeCall* de la trace qui ne connaît pas l'objet source alors que *MethodeMessage* dans diagramme de séquence connaît à la fois l'objet destination et l'objet source. L'extraction des traces consiste à appliquer les règles sur l'instance du méta-modèle du diagramme de séquence. Trois règles de transformations écrites en **OCL** « *Object Constraint Language* » sont appliquées pour le passage des méta-modèles de trace aux méta-modèles du diagramme de séquence. Ces trois règles permettent d'identifier les messages « appel de méthode », les messages « return » et les messages itératifs pour le modèle de diagramme de séquence.

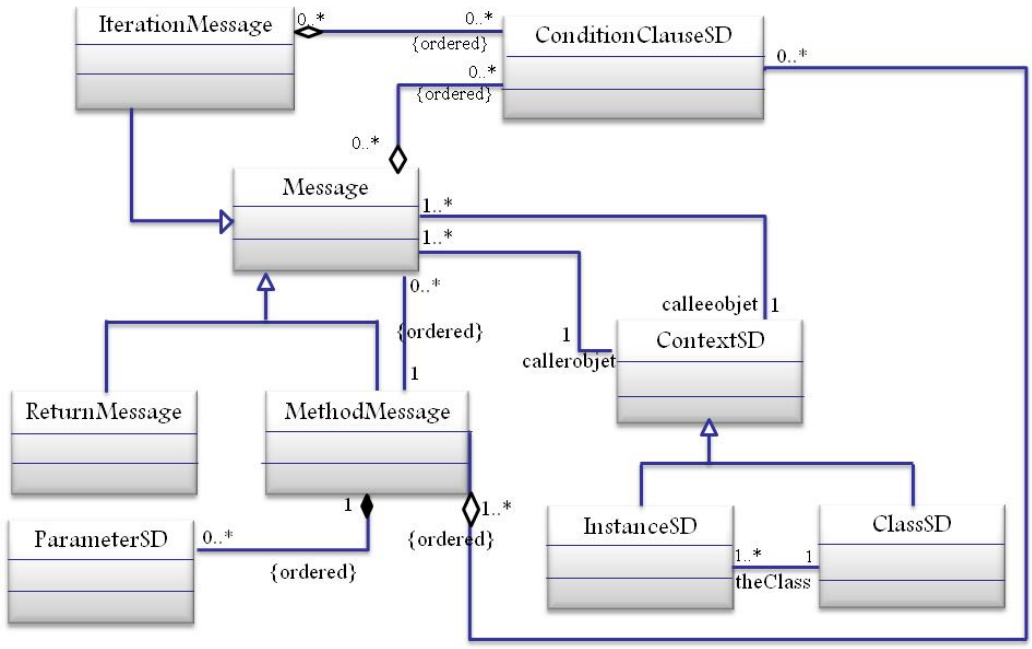


Figure 2.4 : Méta-modèle de diagramme de séquence de Briand et al. [3].

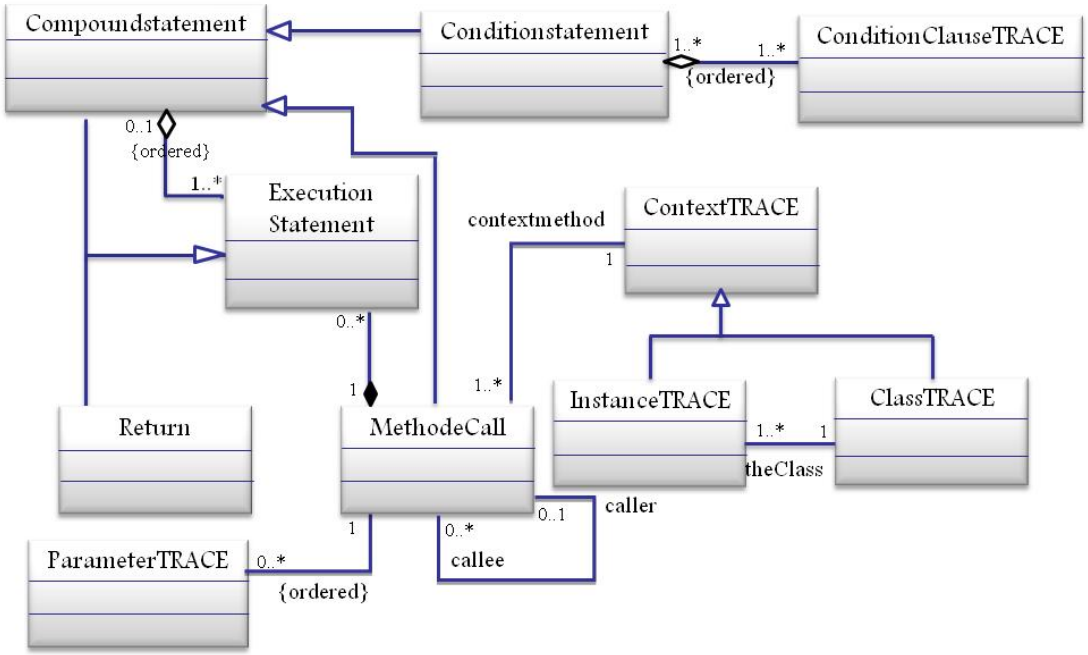


Figure 2.5 : Méta-modèle de trace de Briand et al. [3].

Briand et al. [4] ont étendu leur première approche mentionnée précédemment [3] en détaillant les deux parties suivantes : (1) la récupération des traces d'exécution et la stratégie d'instrumentation, et (2) la stratégie d'enregistrement, les méta-modèles et leurs règles de transformations.

Ils proposent l'instrumentation du code-octet pour éviter le problème de l'instrumentation du code source, en l'occurrence, la gestion de deux versions différentes de code source du programme présentes après instrumentation [4].

Ils emploient la programmation orientée aspect (AOP) et en particulier **AspectJ**[4], visant la récupération des informations suivantes : les objets créés lors de l'exécution du programme, les messages qui circulent entre les différents objets créés lors de l'exécution du programme, les boucles, les conditions et les *threads*. Ces *threads* sont pour différencier entre deux types de messages dans le diagramme de séquence, soient les messages synchrones ou les asynchrones.

Presque comme dans leur ancienne approche [3], ils emploient trois règles de transformations écrites en OCL pour la construction des modèles de diagrammes de séquence à partir des modèles de trace.

Taniguchi et al. [5] proposent une approche automatique pour la rétro-ingénierie des diagrammes de séquence à partir des traces d'exécution d'un programme orienté objet. Ils présentent ces traces sous la forme d'un arbre dont chaque nœud représente un appel de méthode. Ils utilisent quelques règles en plus pour optimiser la taille des traces d'exécution. Nous détaillons dans les paragraphes suivants ces règles en précisant leurs impacts sur l'arbre d'appel ainsi que sur le diagramme de séquence.

---

[4] AspectJ est une extension orientée aspect, créée à Xerox PARC, pour le langage de programmation Java. Cette extension est disponible dans les projets « open-source Eclipse », de manière autonome ou sous forme d'extension pour l'environnement de développement Eclipse. **AspectJ** est devenu le standard du fait de son utilisation répandue pour la programmation orientée aspect en mettant l'accent sur la simplicité et la facilité de mise en œuvre pour les utilisateurs.

- **Règle 1 : Sous-arbres complètement identiques**

Selon la première règle, deux sous-arbres considérés similaires seront remplacés par un seul sous-arbre comme montré dans la Figure 2.6. Une annotation est ajoutée alors à côté pour préciser qu'il s'agit d'une itération.

- **Règle 2 : Sous-arbres différents au niveau des objets**

Comme indiqué dans la Figure 2.7, selon la deuxième règle, deux sous-arbres d'appels considérés similaires, dont les objets ne sont pas identiques, seront remplacés par un seul sous-arbre en ajoutant une notification dans le nœud différent « Objet=2,3 ».

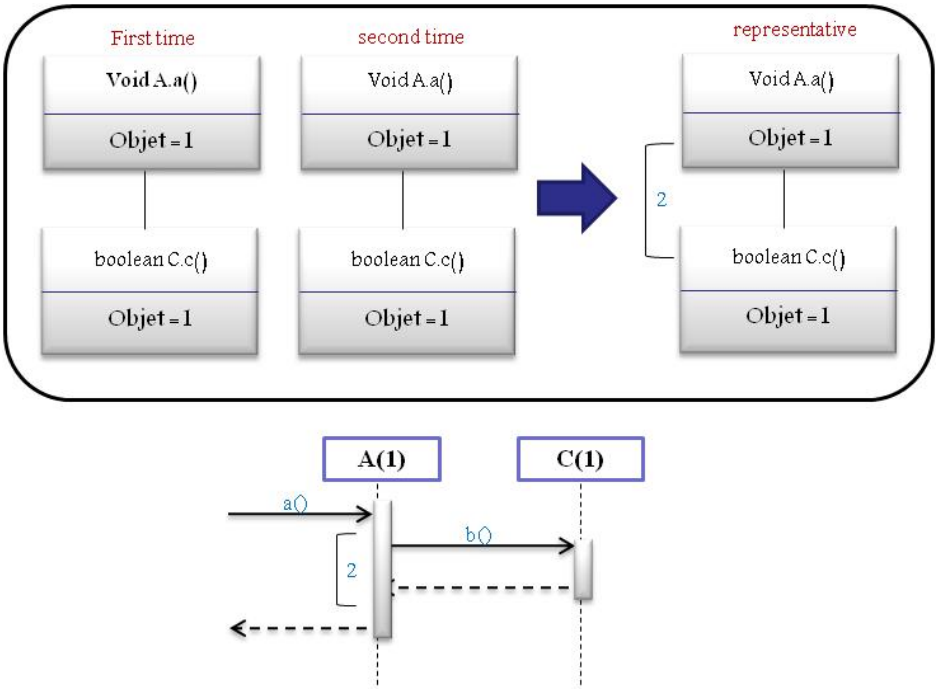


Figure 2.6 : Exemple d'application de la première règle « sous-arbres complètement identiques » de Taniguchi et al. [5].

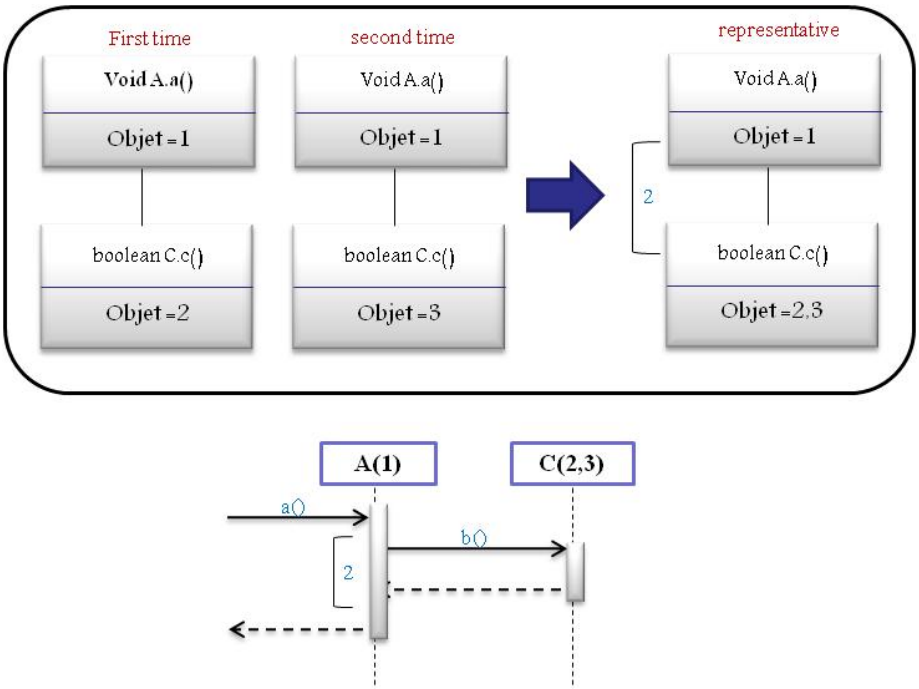


Figure 2.7 : Exemple d'application de la deuxième règle « sous-arbres différents au niveau des objets » de Taniguchi et al. [5].

- **Règle 3 : Manque d'un appel dans un sous-arbre**

Selon la troisième règle, deux sous-arbres considérés similaires, tel qu'un appel manquant dans un sous-arbre, seront remplacés par un seul sous-arbre en gardant le nœud qui diffère (Figure 2.8).

- **Règle 4 : Récursivité**

Comme illustré dans la Figure 2.9, la quatrième règle permet de détecter les appels de méthode récursifs. Ainsi, un appel sera considéré récursif s'il existe une séquence d'appels exécutés successivement de la même méthode avec des objets différents. Cette séquence sera remplacée par une boîte identifiée par « *Rec* + nom de la méthode ».

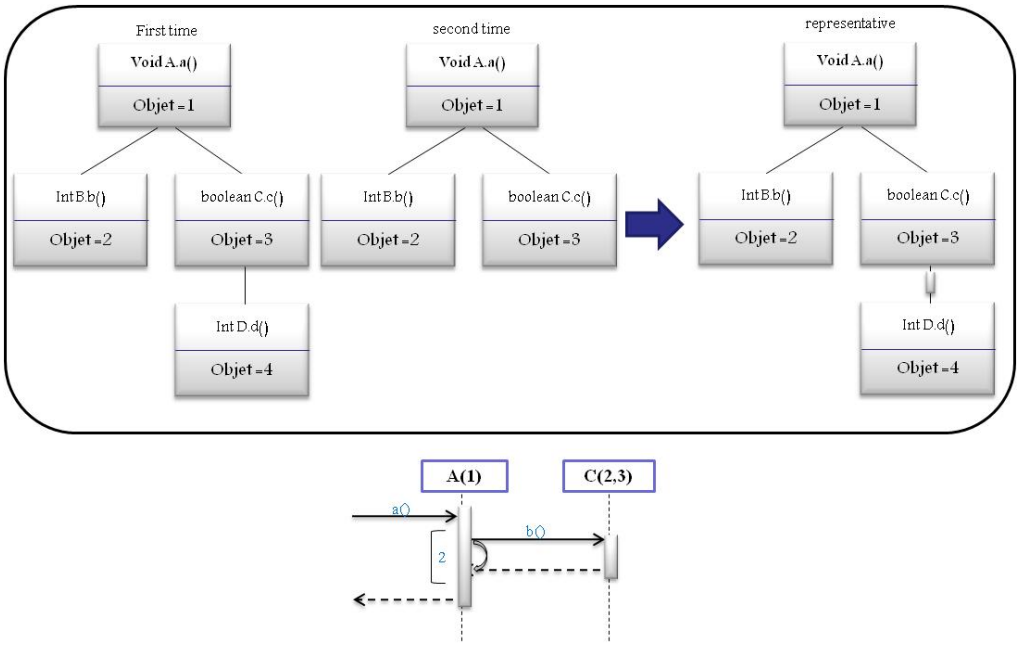


Figure 2.8 : Exemple d'application de la troisième règle « manque d'un appel de méthode » de Taniguchi et al. [5].

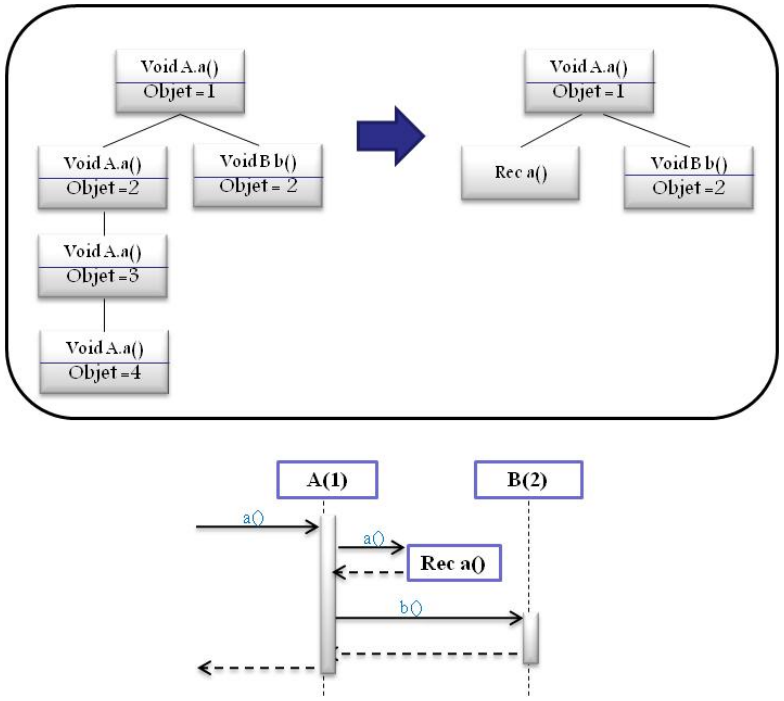


Figure 2.9 : Exemple d'application de la quatrième règle « récursivité » de Taniguchi et al. [5].

Delamaire et al. [6] se concentrent sur l'analyse des traces d'exécution pour la rétro-ingénierie des diagrammes de séquence. L'approche présentée est constituée de deux phases. Durant la première phase, un diagramme de séquence simple est généré contenant juste les appels de méthodes. La deuxième phase permet de tracer des diagrammes de séquence de haut niveau par la combinaison des diagrammes générés dans la première étape en analysant les différents états[5] du système tout en visant à identifier les fragments *alt* et *loop*.

La Figure 2.10 et la Figure 2.11 illustrent un exemple concret qui résume l'approche de Delamaire et al. [6].

Deux diagrammes de séquence simples générés suite à l'exécution d'un même cas d'utilisation sont représentés dans la Figure 2.10, le diagramme combiné est représenté dans la Figure 2.11.

Les messages *m1* et *m3* se répètent dans les deux séquences dont les états initiaux et finaux sont identiques. Les messages *m2* de la première exécution et *m4* de la deuxième exécution ont les mêmes états initiaux, mais pas les mêmes états finaux, et ils sont appelés dans le même niveau. Le problème majeur de cette approche est l'identification des états du système et le choix des éléments des vecteurs. Ce problème s'envenime lorsque des variables négligeables à la compréhension des programmes sont considérées comme des éléments de base, où elles peuvent effectuer des changements au niveau des états du système.

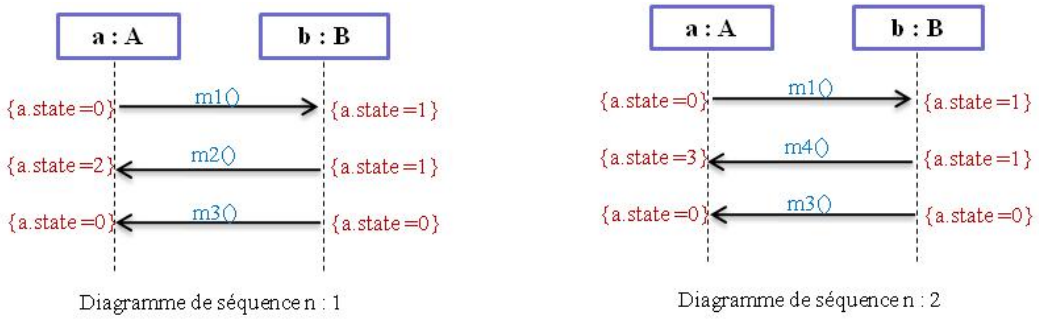


Figure 2.10 : Deux diagrammes de séquence simples générés par Dalamaire et al. [6].

[5] Un état est un vecteur de variables permettant de préciser l'état en-cours du système. Un nouveau vecteur est construit avant et après chaque appel de méthode.

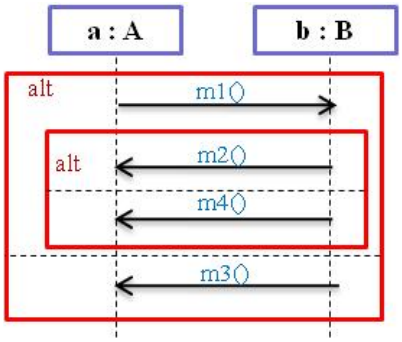


Figure 2.11 : Diagramme de séquence de haut niveau produit par Dalamaire et al. [6].

En étudiant les différentes approches d'analyse dynamique, plusieurs limites peuvent être identifiées. Un des problèmes majeurs des approches automatiques de la rétro-ingénierie des diagrammes de séquence est la grande quantité d'informations extraite rendant la compréhension plus complexe. Ceci nécessite de simplifier les traces d'exécution et d'éliminer les détails inutiles. Ainsi, contrairement aux approches existantes, l'utilisation de techniques de filtrage pour cibler seulement les informations pertinentes est nécessaire. On peut citer principalement trois autres limites :

a) Nommage des objets

Les objets créés lors de l'exécution du système sont identifiés par leurs adresses mémoire. Si l'on utilise ces identifiants dans les diagrammes de séquence, ceci rend la compréhension des diagrammes produits difficile à cause de l'absence de ces identifiants dans le code. De plus, utiliser les adresses pour le nommage des objets diminue la lisibilité pour l'utilisateur.

Comme alternative aux adresses mémoire, on peut utiliser les noms de variables contenant ces objets comme des identifiants. Ceci pose deux problèmes. D'une part, il est nécessaire d'ajouter une analyse statique complexe pour extraire ces noms. D'autre part, un même objet peut être affecté à plusieurs variables. Donc, quelle variable choisir?

Une autre idée consiste à demander à l'usager d'attribuer des noms aux objets, mais ceci est impossible dans une approche totalement automatique.

b) Une seule exécution n'offre pas une vue complète des cas d'utilisation

Dans plusieurs cas des approches existantes, les diagrammes extraits correspondent à des exécutions uniques. Or pour générer un bon diagramme de séquence, il est nécessaire d'exécuter plusieurs fois le même cas d'utilisation, mais de différentes façons, et de les subdiviser pour couvrir toutes les possibilités.

c) Polymorphisme

Le problème de polymorphisme survient lors d'une boucle d'appels sur un type abstrait. Le premier diagramme de séquence présenté dans la Figure 2.12 est généré suite à une analyse dynamique des traces d'exécution: un message *m* est envoyé trois fois par le même objet « *a : A* » vers trois objets différents. Par contre, en analysant la structure statique de ces trois objets (diagramme de classe), on trouve que les trois classes héritent d'une classe abstraite « *B* ». Le problème qui se pose est que les approches existantes ne permettent pas une simplification du diagramme en détectant les liens de généralisation, ce qui pourrait optimiser le diagramme de séquence généré.

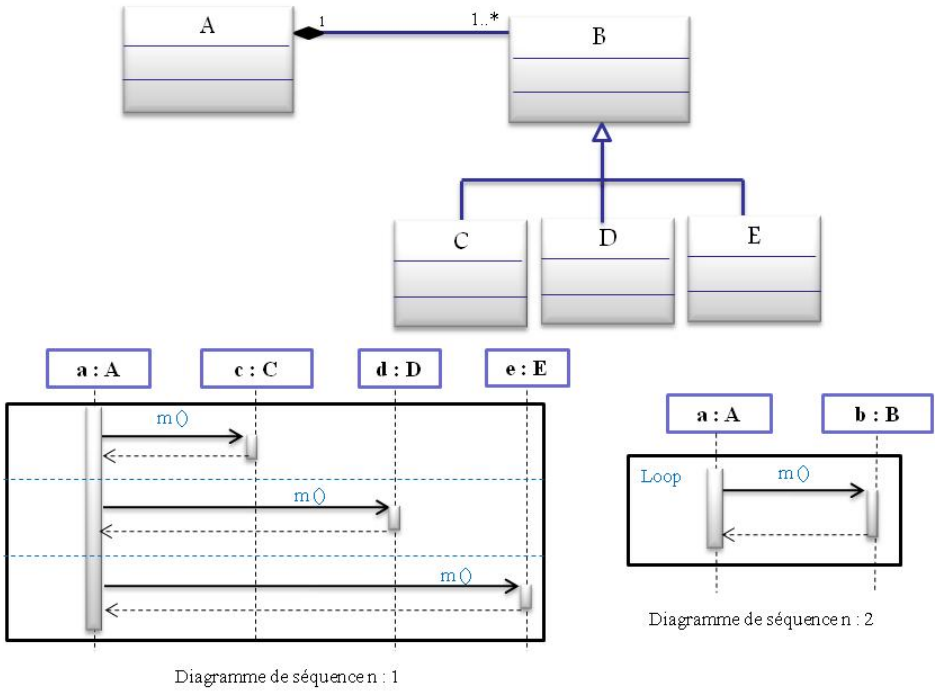


Figure 2.12 : Exemple d'un problème causé par le polymorphisme pour la rétro-ingénierie des diagrammes de séquence.

Notre solution alternative consiste à proposer une approche semi-automatique basée sur les techniques de visualisation pour combler les limites citées ci-dessus.

Nous proposons dans la sous-section suivante un aperçu des concepts et des travaux reliés à la visualisation.

2.3. Visualisation

2.2.1. Visualisation des logiciels

En informatique et en particulier dans le domaine du génie logiciel, la visualisation est parfois employée pour l'analyse des logiciels. On peut distinguer entre deux paradigmes : la visualisation de la structure des logiciels et la visualisation du comportement des logiciels.

a) Visualisation de la structure des logiciels

La visualisation de la structure consiste à visualiser les éléments constituant le logiciel (packages, classes, interfaces), ainsi que les relations entre eux. Eick et al. [12] ont développé l'outil «*SeeSoft*» qui permet de visualiser des statistiques concernant les lignes du code d'un logiciel. *SeeSoft* donne une vue d'ensemble du code d'un logiciel en représentant chaque ligne de code par une barre horizontale. Ces barres sont ensuite organisées verticalement similairement aux instructions du programme. Pour réduire au maximum la place nécessaire à leur représentation, la hauteur de chaque barre horizontale est réduite à un seul pixel. La couleur de chaque ligne est assignée selon une valeur statistique, par exemple, la date à laquelle la ligne a été ajoutée au code. Une capture d'écran de leur outil est présentée dans la Figure 2.13.

Wettel et al. [11] utilisent une métaphore de la ville pour représenter le logiciel. Les entités du logiciel, représentées par des boîtes 3D, sont distribuées sur un plan 2D selon l'architecture de bas niveau du programme. Les « édifices » sont caractérisés par cinq indices visuels : dimensions, position, couleur, saturation et transparence. Chaque indice visuel correspond à une métrique de l'entité logicielle, représentée par l'édifice (Figure 2.14).

b) Visualisation du comportement des logiciels

La visualisation du comportement consiste à visualiser les traces d'exécution du logiciel grâce à différentes métaphores. De Pauw et al. [7] décrivent leur stratégie pour l'organisation et la visualisation des informations obtenues lors de l'exécution d'un programme orienté objet. Leur approche consiste à respecter les trois critères suivants :

- ***Perfection :*** extraction d'autant d'informations que possible sur les aspects significatifs de l'exécution d'un programme en réduisant au minimum les pertes d'informations.
- ***Compacité :*** utilisation du minimum de stockage pour contenir les informations capturées.
- ***Récupération efficace :*** arrangement des informations de séquence en assurant un accès facile et rapide.

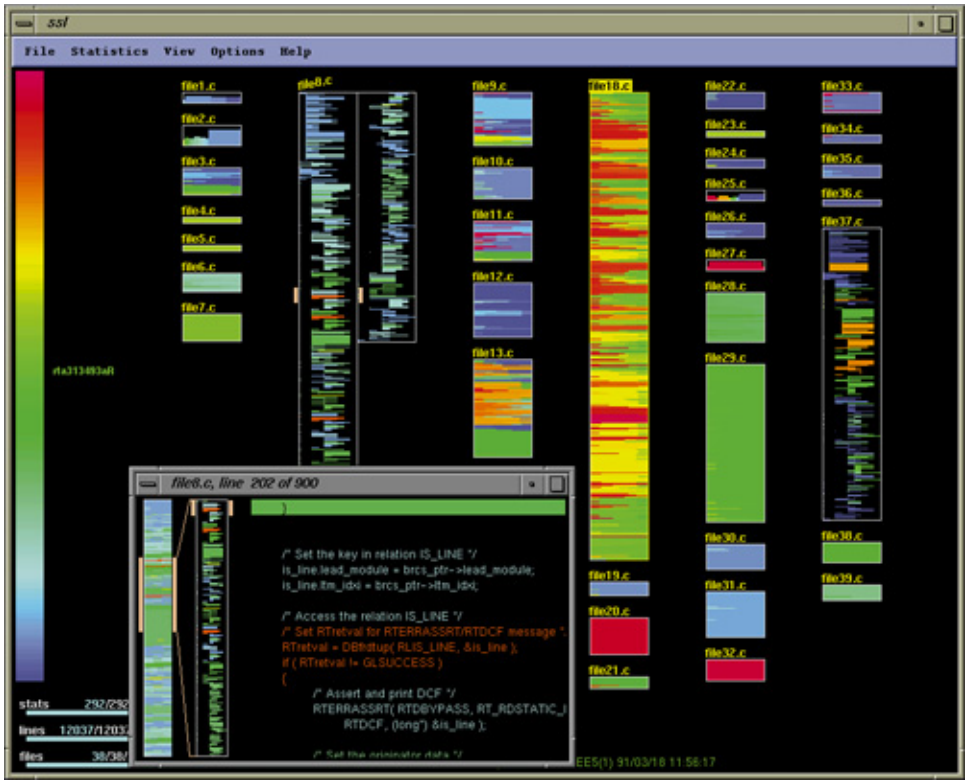


Figure 2.13 : Un exemple d'utilisation de *SeeSoft* de Eick et al. [12].

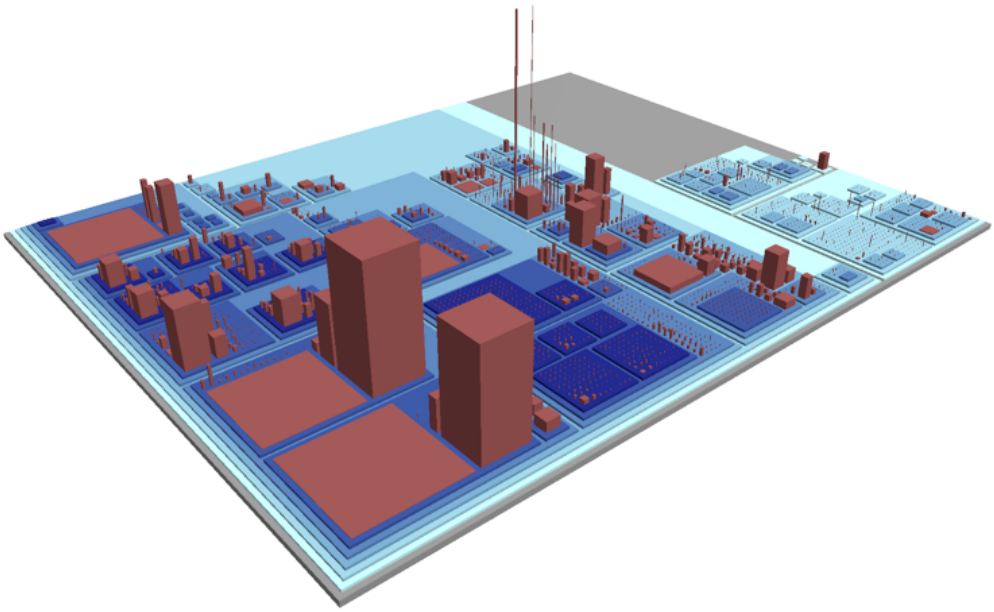


Figure 2.14 : Un exemple d'utilisation de *CodeCity* de Wettel et al. [11].

Comme illustré dans la Figure 2.15, leur approche se base sur l'organisation de l'information obtenue (les traces d'exécution) dans un espace à quatre dimensions : *Classes*, *Objets*, *Méthodes* et *Temps*. Chaque point 4D correspond à un évènement qui peut être par exemple ***une construction*** d'un objet via une méthode de construction, ***une destruction*** d'un objet via une méthode de destruction ou une méthode d'achèvement, ***un événement d'entrée*** qui marque un message envoyé contenant la méthode et ses paramètres, etc.

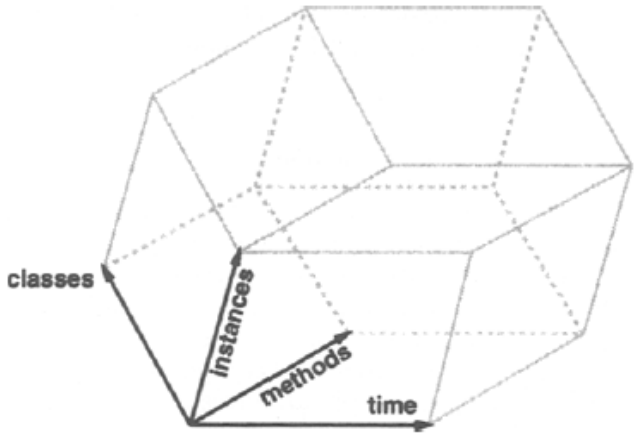


Figure 2.15 : Modèle d'organisation des traces par De Pauw et al. [7].

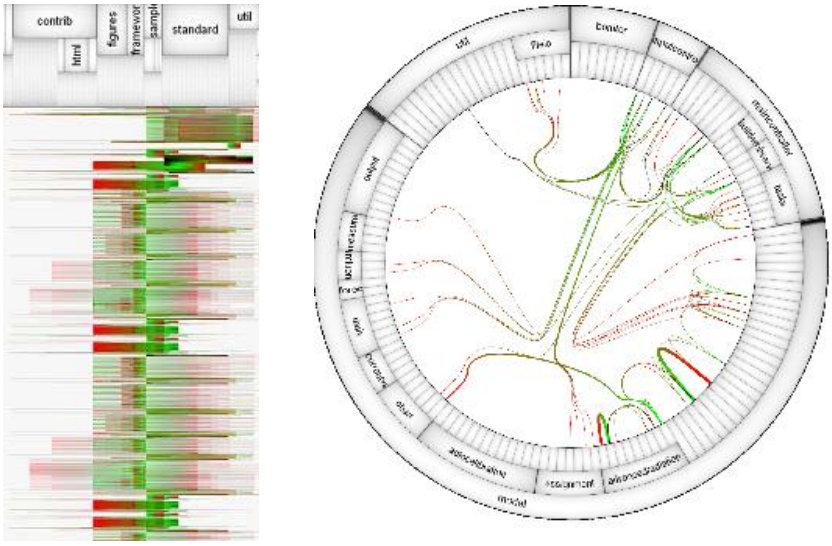


Figure 2.16 : Un exemple d'utilisation de *EXTRAVIS* pour la visualisation des traces d'exécution de Cornelissen et al. [8].

Cornelissen et al. [8] proposent une étude basée sur la conception d'une expérience contrôlée pour évaluer l'utilité de la visualisation sur la compréhension des traces des programmes, en validant leur outil pour valider *EXTRAVIS* (Figure 2.16).

Pour mieux comprendre la nature de la valeur ajoutée par *EXTRAVIS*, ils identifient les différents types de tâches qui profitent le plus aux traces d'exécution.

Ils décrivent une expérience dans laquelle ils mesurent la façon dont leur outil améliore le temps nécessaire pour la compréhension des types de tâches et l'exactitude des réponses données au cours de ces tâches.

2.2.2. Visualisation interactive

Par souci d'efficacité, la visualisation de données doit être interactive, en permettant d'affiner dynamiquement l'affichage et de répondre aux requêtes graphiques de l'utilisateur. La visualisation interactive est la combinaison des techniques de visualisation avec les techniques d'interaction. Diverses techniques ont été développées afin de faciliter la compréhension globale et la recherche d'informations dans un environnement visuel.

Zhou et al. [13] ont identifié un ensemble de tâches visuelles qui servent d'interfaces entre les buts d'un utilisateur (*intents*) et les techniques de visualisation. Leur approche consiste à construire automatiquement des présentations multimédias dans leur outil *IMPROVISE*. Ces constructions sont basées sur les intérêts et les objectifs de l'utilisateur.

Shneiderman et al. [14] présentent une étude décrivant comment l'utilisateur interagit avec la visualisation. Ils détaillent un ensemble de techniques associées à la visualisation d'informations comme les filtrages dynamiques, les zooms spécifiques, les transformations sémantico-géométriques, etc. Selon l'approche de Shneiderman et al., la perception est indissociable de l'action, et donc il faut agir pour percevoir, et il faut percevoir pour agir. Il s'agit d'une boucle *action-perception*.

2.4. Conclusion

Nous remarquons que d'une part, les approches d'extraction des diagrammes de séquences ont de la difficulté à filtrer l'information. D'autre part, les techniques de visualisation interactive permettent à un usager d'agir efficacement sur des grands ensembles de données. Nous proposons donc comme alternative à l'automatisation, d'utiliser les techniques de visualisation interactive pour extraire des diagrammes de séquence plus utiles à la compréhension des programmes et plus compacts. Notre approche permet aux analystes d'utiliser leurs connaissances contextuelles pour filtrer des détails inutiles. Ceci fera l'objet du prochain chapitre.

Chapitre 3

Approche proposée

3.1. Introduction

Dans le cadre de notre projet de maîtrise, nous proposons un outil pour la rétro-ingénierie des diagrammes de séquence en nous basant sur l'analyse dynamique des traces d'exécution et sur les techniques de visualisation interactives.

Dans ce chapitre, nous exposons notre approche pour la rétro-ingénierie des diagrammes de séquence, nous commençons par donner un schéma global de l'approche, et ensuite nous détaillons les différentes phases de notre travail selon trois volets : (1) la génération des traces d'exécution, (2) la combinaison des traces d'exécution, et (3) la visualisation et la navigation à travers les traces d'exécution pour l'extraction des diagrammes de séquence. Enfin, nous terminons ce chapitre par une conclusion.

3.2. Approche globale

3.2.1. Exemple de scénario

Afin de mieux décrire notre approche, nous commençons cette section en donnant un exemple de scénario que nous utiliserons tout au long de ce chapitre. Notre exemple est un scénario d'utilisation d'un outil de dessin[6]. Il s'agit d'une application Java de 16 classes permettant la création de dessins à l'aide d'une interface graphique.

Le scénario retenu permet de dessiner des représentations géométriques. Trois exécutions de ce scénario sont effectuées pour la construction respectivement d'un rectangle, d'un cercle et d'un segment. Dans chaque cas, l'utilisateur fournit des informations telles que la couleur de remplissage, la couleur de la bordure, la taille, etc.

[6] http://www.javafr.com/codes/JavaDessin_36623.aspx

3.2.2. Schéma global de notre approche

La rétro-ingénierie des modèles de comportement se réalise pour plusieurs raisons, telles que l'analyse, la compréhension et la documentation du code. Notre travail est motivé par la redocumentation d'une application existante. On extrait les diagrammes de séquence pour des cas d'utilisations en se basant sur l'analyse dynamique. Notre approche est constituée de trois étapes illustrées dans la Figure 3.1.

D'abord, on génère un ensemble de traces d'exécution correspondant à un scénario d'exécution spécifique. Les traces sont obtenues par l'instrumentation et l'exécution du code. Nous utilisons plusieurs traces d'exécution afin de mieux capturer les variations d'un même scénario.

Ensuite, nous alignons les traces par l'identification de fragments d'exécution communs et les situations spécifiques à chaque exécution. Notre algorithme d'alignement permet également d'identifier les comportements qui pourraient être extraits des fragments semblables de différentes exécutions. Notre algorithme est basé sur la méthode d'alignement des séquences de caractères de Smith et Waterman [16].

La troisième étape de notre approche consiste à extraire le diagramme de séquence en utilisant un système de visualisation interactif. Cet environnement permet la navigation dans la trace combinée et l'exécution des opérations d'extraction.

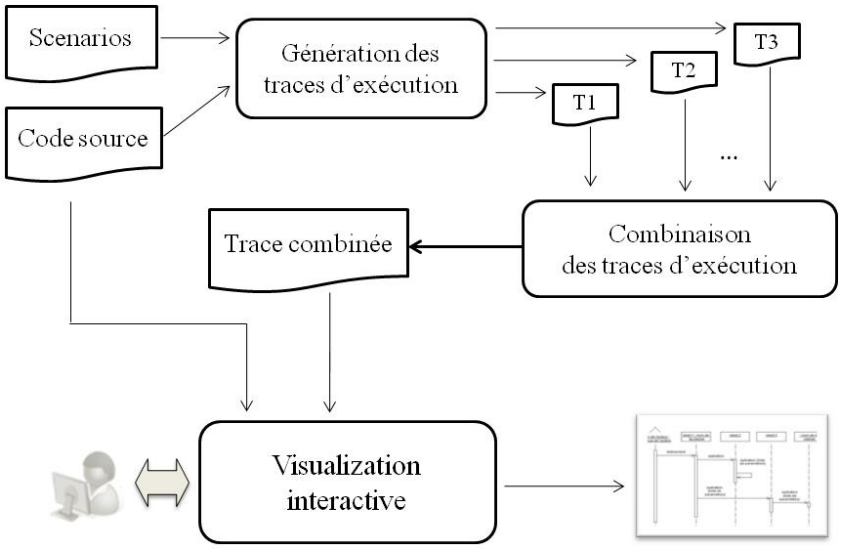


Figure 3.1 : Schéma global de notre approche.

Nous détaillons dans ce qui suit les différents composants de notre approche.

3.3. Génération et alignement des traces d'exécution

3.3.1. Génération des traces d'exécution

Cette étape consiste à récupérer les informations nécessaires relatives à chaque exécution. Il s'agit de collecter des traces à partir du code. Dans le cadre de notre travail, nous instrumentons le code pour récupérer toutes les informations pertinentes concernant les objets instanciés de différentes classes du code ainsi que tous les appels de méthodes, et les structures conditionnelles et itératives. Comme notre objectif est de documenter les scénarios et les cas d'utilisations existants, nous faisons l'hypothèse que l'extraction des diagrammes de séquence se fait à partir des scénarios déjà existants. Pour un scénario particulier, nous déterminons l'ensemble des exécutions qui aident à capturer les variations dans le scénario. Dans notre exemple, le scénario consiste à dessiner une figure géométrique avec une application de dessin. Nous sélectionnons trois exécutions différentes correspondant à trois formes géométriques : rectangle, cercle et segment. Ces formes ont été choisies parce qu'elles ont des propriétés différentes, ce qui altère leurs traces respectives.

Avant de commencer l'exécution, le code est instrumenté pour extraire les informations nécessaires. Pour cela, nous injectons des notifications d'état pour le début et la fin de chaque méthode, de chaque bloc d'une boucle, et de chaque bloc d'une condition. Ces notifications permettent de déterminer où chaque appel de méthode est exécuté et comment les appels sont organisés. Par conséquent, on obtient une séquence d'appels de méthodes, où chaque appel de méthode contient les informations suivantes :

- Le type et l'adresse mémoire de l'objet expéditeur de message.
- Le type et l'adresse mémoire de l'objet receveur de message.
- Le nom de la méthode ainsi qu'un identificateur d'exécution de la méthode appelante. Cet identificateur permet de différencier entre deux exécutions différentes de la même méthode.
- Le nom de la méthode et l'identificateur d'exécution de la méthode appelée.

- La pile des blocs conditionnels et itératifs où l'appel de méthode est exécuté.

La représentation de chaque appel de méthode dans la trace suit le format suivant :

```
<type_objet_expéditeur> "["<adresse_objet_expéditeur>"]", <type_objet_
receveur> "["<adresse_objet_ receveur>"]", <nom_méthode_appelante>
"["<id_exécution_méthode_appelante>"]", <nom_méthode_appelée>
"["<id_exécution_méthode_appelée>"]",
"<"<pile_blocs_conditionnels_iteratifs>">"
```

```
1   , PanelDraw [21668571],  , StartDraw [T1M1],  ;
2  PanelDraw [21668571], Figure [3916193], StartDraw
   [T1M1], Figure [T1M2], _
3  Figure [3916193], State [23930626], Figure [T1M2],
   getTransparency [T1M3], _
4  Figure [3916193], State [23930626], Figure [T1M2],
   getMode [T1M4], _
5  Figure [3916193], State [23930626], Figure [T1M2],
   getLineColor [T1M5], _
6  Figure [3916193], State [23930626], Figure [T1M2],
   getThickness [T1M6], _
7  Figure [3916193], State [23930626], Figure [T1M2],
   getFillingColor [T1M7], _
8  Figure [3916193], State [23930626], Figure [T1M2],
   getFigureType [T1M8], _
9  PanelDraw [21668571], Circle [17282414], StartDraw
   [T1M1], Circle [T1M9],
   <%(State.getFiguretype()==MODE_CERCLE)%>
10 Circle [17282414], Figure [3916193], Circle
   [T1M9], draw [T1M10],
   <%(State.getFiguretype()==MODE_CERCLE)%>
11 PanelDraw [21668571], Figure [3916193], StartDraw
   [T1M1], insert [T1M11], _
```

Figure 3.2 : Fragment de la trace d'exécution pour le dessin d'un cercle.

```
1   , PanelDraw [64836544],  , StartDraw [T2M1],
2  PanelDraw [64836544], Figure [9826352], StartDraw
   [T2M1], Figure [T2M2], _
3  Figure [9826352], State [19286486], Figure
   [T2M2], getTransparency [T2M3], _
4  Figure [9826352], State [19286486], Figure
   [T2M2], getMode [T2M4], _
5  Figure [9826352], State [19286486], Figure
   [T2M2], getLineColor [T2M5], _
6  Figure [9826352], State [19286486], Figure
   [T2M2], getFillingColor [T2M6], _
7  Figure [9826352], State [19286486], Figure
   [T2M2], getFigureType [T2M7], _
8  PanelDraw [64836544], Rectangle [1271231],
   StartDraw [T2M1], Rectangle [T2M8],
   <%(State.getFiguretype()==MODE_RECTANGLE)%>
9  Rectangle [1271231], Figure [9826352], Rectangle
   [T2M9], draw [T2M9],
   <%(State.getFiguretype()==MODE_RECTANGLE)%>
10 PanelDraw [64836544], Figure [9826352], StartDraw
   [T2M1], insert [T2M10], _
```

Figure 3.3 **:** Fragment de la trace d'exécution pour le dessin d'un rectangle.

La Figure 3.2 montre un exemple de trace d'exécution correspondant à la construction d'un cercle. Le premier événement correspond à des exécutions de la première méthode `StartDraw` par l'objet `PanelDraw` de l'adresse mémoire `21668571`. Pour cet appel, il n'y a aucun objet appelant ni de méthode appelante. Quant à l'absence de bloc itératif ou conditionnel, cette information inexistante est représentée par le caractère `"_"`.

La méthode `StartDraw` fait appel à trois méthodes : le constructeur `Figure` (événement 2), le constructeur `Circle` (événement 9) et la méthode `insert` (événement 11). Pour différencier les deux exécutions de la même méthode, l'exécution de `StartDraw` est identifiée par [T1M1] qui est utilisé dans les trois appels (événements 2, 9 et 11). Enfin, les méthodes `Circle` et `draw` des événements 9 et 10 sont appelés à l'intérieur de l'exécution du bloc conditionnel suivant :

(`State.getFiguretype()==MODE_CERCLE`).

3.3.2. Alignement des traces d'exécution

Pour avoir une vue globale et complète du système informatique pour un cas d'utilisation précis, il est nécessaire d'analyser plusieurs exécutions du même scénario

dont les données d'entrée sont différentes. Nous nous intéressons dans cette deuxième phase à combiner les différentes traces d'exécution.

En effet, les scénarios de cas d'utilisations définissent en général les possibilités abstraites d'exécution d'un système particulier. Un même scénario pourrait avoir plusieurs variations différentes qui produisent des traces d'exécution différentes, quoique semblables. Dans notre exemple, il n'est pas réaliste d'avoir un scénario pour chaque type de forme géométrique. Pour documenter le programme, un scénario unique qui consiste à dessiner une forme géométrique est suffisant. Pour cette raison, il est intéressant d'exécuter le scénario avec des variations différentes. La Figure 3.3 montre un exemple de la trace correspondant au dessin d'un rectangle. Lorsqu'un rectangle est instancié au lieu d'un cercle, la méthode `getThickness` n'est pas appelée parce que la valeur d'épaisseur n'a pas été fournie dans la boîte de dialogue.

Pour aider à identifier les comportements communs et les différences entre les exécutions d'un même scénario, nous alignons les traces correspondantes. Lorsqu'il s'agit de plus de deux traces à combiner, la combinaison se fait progressivement. Dans notre exemple, nous alignons les deux traces correspondant à la construction d'un rectangle et d'un cercle, et par la suite, la trace résultante est alignée avec la trace correspondant à la construction d'un segment. Comme les traces d'exécution sont représentées par des arbres où chaque nœud est un appel de méthode, notre algorithme d'alignement met en œuvre un processus récursif, où les arbres sont comparés niveau par niveau. Deux nœuds sont alignés s'ils ont les mêmes types respectivement des objets expéditeurs et récepteurs, les mêmes noms respectivement pour les méthodes appelantes et appelées, et la même pile de conditions et de boucles. En outre, deux nœuds peuvent être comparés si leurs parents respectifs sont déjà alignés. L'algorithme commence par comparer les nœuds racines des deux traces. Si les deux nœuds sont alignés, alors leurs nœuds fils sont comparés. Ensuite, l'algorithme est appliqué récursivement à chaque paire de nœuds alignés.

Dans chaque trace, les nœuds fils représentent la méthode des appels effectués au sein de la méthode du nœud parent. Comme les appels sont ordonnés, ils définissent une

séquence. Pour comparer les séquences des nœuds fils dans les deux traces, nous utilisons l'algorithme *Smith-Waterman* [16].

- Algorithme *Smith-Waterman*

Cet algorithme est basé sur la programmation dynamique. Il explore tous les alignements possibles entre deux chaînes de caractères, en utilisant des matrices de substitution pour générer un alignement optimal entre les deux séquences de caractères. Nous avons adapté cet algorithme pour aligner des séquences d'appels.

L'algorithme *Smith-Waterman* s'exécute récursivement pour stocker la valeur de similarité des sous-séquences. Lors de l'alignement des deux séquences ($a_1, ..., a_n$) et ($b_1, ..., b_m$), nous définissons une matrice de substitution M ayant $n+1$ lignes et $m+1$ colonnes. Chaque position $M_{i,j}$ correspond à la meilleure valeur de l'alignement tenant compte des éléments des séquences précédemment alignés. L'algorithme peut introduire des "gaps" (représenté dans l'exemple ci-après par "_") afin d'améliorer la similarité entre les séquences.

Formellement,

$$M_{i,0} = M_{0,j} = 0$$

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + match(a_i, b_j) \\ M_{i-1,j} + gap(a_i, \_) \\ M_{i,j-1} + gap(\_, b_j) \end{cases}$$

match (a_i,bj) définit la valeur de similarité entre les deux caractères a_i et b_j. Lorsque le gap est inséré avant a_i (resp. b_j), il encourt une pénalité d'insertion d'un *gap* (a_i, _) (resp. *gap* (_, b_j)).

Notre adaptation de l'algorithme d'alignement est directe. Nous mettons simplement la pénalité d'insertion d'un gap à -1 et nous utilisons notre nœud (l'appel de méthode) pour mesurer la valeur de similarité. Ainsi, pour deux nœuds a_i et b_j, le *match* (a_i, b_j) est mis à 1 si a_i et b_j pourraient être alignés.

L'alignement des traces T1 et T2 (la Figure 3.2 et la Figure 3.3) est présenté à la Figure 3.4. Par souci de concision, nous identifions les nœuds par :

"a" <numéro de trace>"-"< numéro d'événement >.

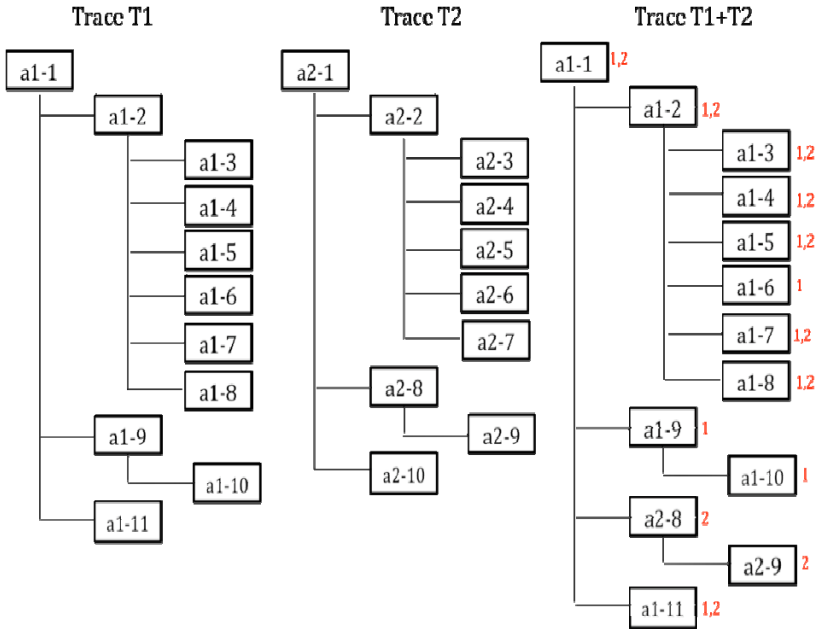


Figure 3.4 : Exemple de combinaison de traces.

Dans la trace combinée, les nœuds sont étiquetés par les numéros de traces où ils apparaissent. Par exemple, le nœud *a1-3* [1,2] dans T1+T2 indique que les deux appels de méthode T1 (*a1-3*) et T2 (*a2-3*) ont été alignés. Comme les adresses mémoire diffèrent d'une trace à une autre, nous gardons selon la convention celles de la première trace, ce qui explique pourquoi nous réutilisons les noms de nœuds de T1 dans T1+T2.

Pour illustrer l'alignement des séquences des nœuds fils, nous considérons des méthodes *a1-3* à *a1-8* appelées par *a1-2* dans T1 et des méthodes *a2-3* à *a2-7* appelées par *a2-2* dans T2. Ces deux listes peuvent être alignées parce que *a1-2* et *a2-2* sont déjà alignés. La Figure 3.5 montre le meilleur alignement possible entre ces deux séquences. Un gap unique est inséré dans T2 pour prendre en considération l'absence de l'appel `getThickness`.

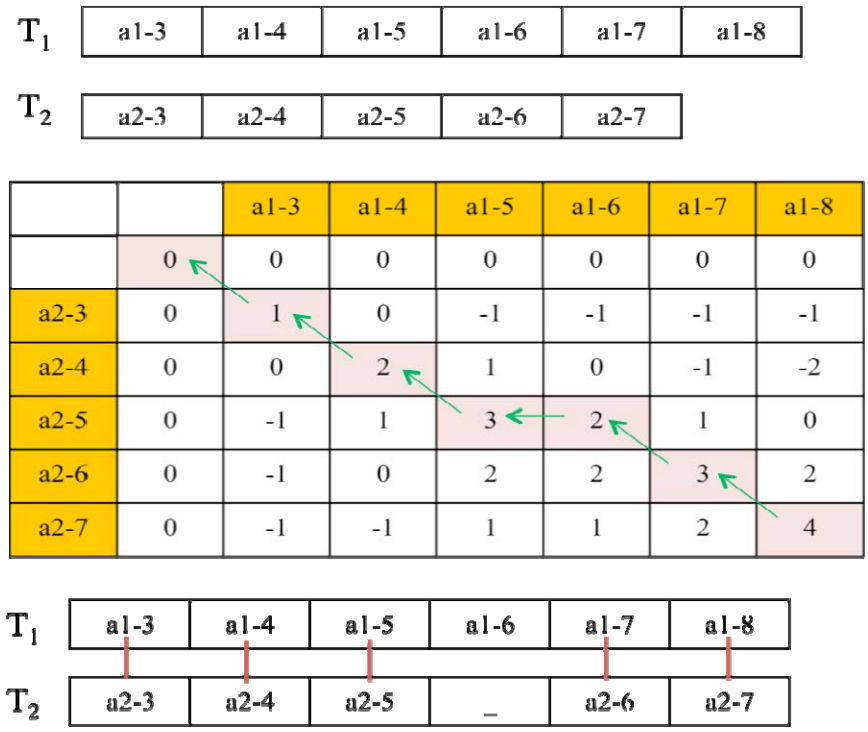


Figure 3.5 : Alignement des séquences de nœuds fils.

- Recommandation

Notre stratégie de combinaisons ne permet pas la fusion automatique des sous-arbres dont les racines (appels) font intervenir des objets appelés de classes différentes, mais qui héritent d'une même super classe. Notre algorithme d'alignement permet de détecter ce type de situations, mais laisse la décision finale à l'utilisateur lors du processus de génération des diagrammes de séquence. En effet, comme nous l'expliquons dans la Section 3.4.3.4, ces alignements potentiels sont suggérés à l'utilisateur via un système de recommandations.

3.4. Visualisation et navigation des traces pour la rétro-ingénierie des diagrammes de séquence

Nous traitons dans cette section la deuxième partie de notre projet où nous employons des métaphores de visualisation et d'interaction pour la rétro-ingénierie des diagrammes de séquence. En effet, nous détaillons le processus d'extraction du

diagramme de séquence. Par la suite et avant l'explication de l'environnement de visualisation, les actions de navigation et les interactions possibles, nous donnons les transformations de base de la trace combinée vers le diagramme de séquence.

3.4.1. Processus d'interaction

Le processus d'interaction est un ensemble de cycles successifs qui contiennent des transformations automatiques et des interactions avec l'utilisateur pour modifier et compléter ces transformations initiales. Chaque cycle représente une étape du parcours et du traitement de la trace combinée. Au début du cycle, l'appel courant est visualisé dans la vue globale (Section 3.4.2.1). En suite, les transformations de base (Section 3.4.2) sont appliquées à cet appel. Le résultat des transformations est ajouté dans la vue diagramme (Section 3.4.2.2). Si des recommandations de fusion (Section 3.4.3.4) ont été détectées lors de l'alignement, celles-ci sont suggérées à l'utilisateur. La dernière étape du cycle consiste à donner la main à l'utilisateur pour effectuer les interactions suggérées dans la section 3.4.4.

Dans ce cycle d'interaction, l'utilisateur dispose des vues sur la trace combinée et le diagramme généré. C'est la présence des deux vues qui permet à l'utilisateur d'accepter/refuser les transformations de base et les recommandations. Elle lui permet aussi d'effectuer des opérations de base telles que le renommage. De plus, la vue diagramme étant limitée par la taille de l'écran (seul un fragment du diagramme est visible à la fois sans défilement), une vue globale efficace permet de situer le contexte de ce fragment dans le scénario global.

3.4.2. Transformation de base des traces vers un diagramme de séquence

En explorant la trace combinée, chaque appel de méthode est transformé par défaut comme suit :

- Un appel de méthode est transformé en un message dans le diagramme de séquence.

- La source d'un message est la ligne de vie de l'acteur correspondant à l'objet de la méthode où l'appel s'est produit.
- Si l'objet appelé existe déjà en tant qu'acteur dans le diagramme, le message sera lié à sa ligne de vie, sinon un nouvel acteur sera créé tel que son d'identité est l'adresse mémoire de l'objet appelé, et le message connecte à sa ligne de vie.
- Les boîtes opt/alt/boucle sont dessinées de façon progressive pour encapsuler des messages en fonction des piles de conditions/boucles associées à leurs appels de méthode.
- Les messages de retour sont insérés dans le graphique en fonction de la structure de l'arbre de la trace combinée. Ces messages connectent les acteurs respectivement récepteurs et expéditeurs des méthodes associées.

3.4.3. Vues interactives

À chaque étape du processus d'extraction du diagramme de séquence, l'analyste est capable de visualiser le fragment déjà traité de la trace combinée (vue globale) et son fragment correspondant du diagramme de séquence (vue diagramme). Dans les paragraphes suivants, nous détaillons ces deux vues.

3.4.2.1. Vue globale

La taille des traces d'exécution rend leur visualisation plus difficile. Cependant, nous croyons que l'analyste prend de meilleures décisions quand il peut afficher les traces, au moins partiellement. Par conséquent, nous avons décidé de visualiser la trace combinée progressivement à la navigation.

a) Représentation des méthodes

Les méthodes sont représentées par des cylindres disposés sur un plan qui est divisé en une structure de type grille. Chaque cellule de la grille peut héberger au maximum un seul cylindre. Les méthodes sont colorées en gris. La méthode contrôlée à l'exécution est colorée en bleu.

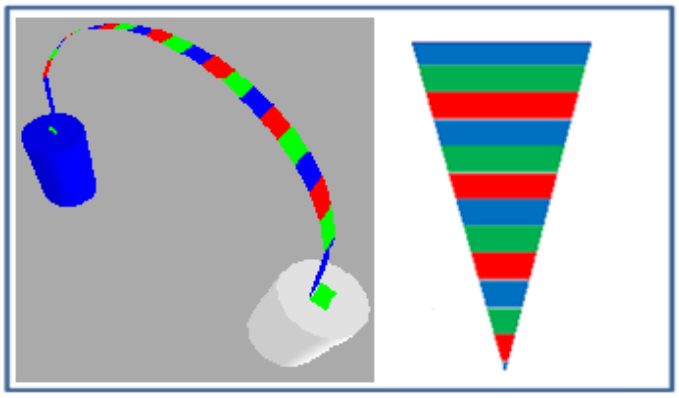
Nous représentons un appel de méthode par un arc sous forme d'un demi-cercle partant du cylindre de la méthode appelante vers le cylindre de la méthode appelée comme le

montre la Figure 3.6 (a). Cette représentation des arcs avec différentes hauteurs facilite la perception des appels en réduisant les croisements.

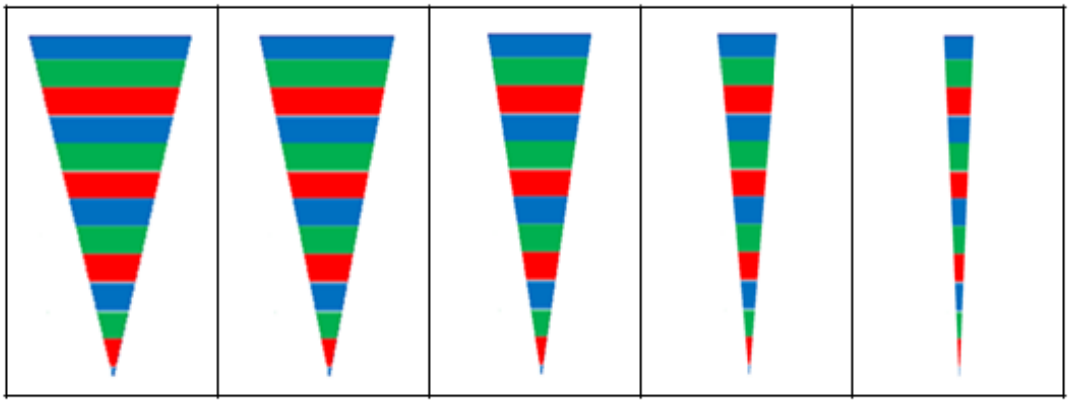
La représentation de l'arc indique également le sens de l'appel et les traces où elle apparaît. Les directions sont représentées par des triangles isocèles dont la base est attachée à la méthode appelante et l'extrémité du sommet est attachée à la méthode appelée. Le triangle est divisé en bandes alternantes colorées en fonction des traces (une couleur par trace). Dans la Figure 3.6(a), la coloration de l'arc indique que l'appel est présent dans les trois traces correspondantes respectivement aux couleurs bleu, vert et rouge.

Bien que nous représentions la trace combinée progressivement, la vue peut devenir encombrée, surtout après avoir parcouru une importante partie de la trace. Pour aider à concentrer l'analyste sur les appels les plus récents tout en gardant le contexte global, nous réduisons progressivement la largeur des arcs suite à l'augmentation de l'âge des appels correspondants. Nous déterminons l'âge d'un appel selon le nombre d'appels qui le sépare par rapport à l'appel en cours. La réduction de la largeur est illustrée dans la Figure 6 (b).

Les retours de méthodes sont également représentés par des arcs dont la largeur est réduite avec l'âge (Figure 3.7). La seule différence est que nous utilisons un triangle en pointillé avec une couleur unique pour aider l'analyste à distinguer entre les appels et les retours. Comme un appel et son retour correspondant sont entre les mêmes méthodes (mais dans des directions opposées), il n'est pas nécessaire d'indiquer les traces où ils apparaissent.



(a)



(b)

Figure 3.6 : Représentation des appels de méthodes: (a) représentation initiale et (b) réduction de largeur selon âge.

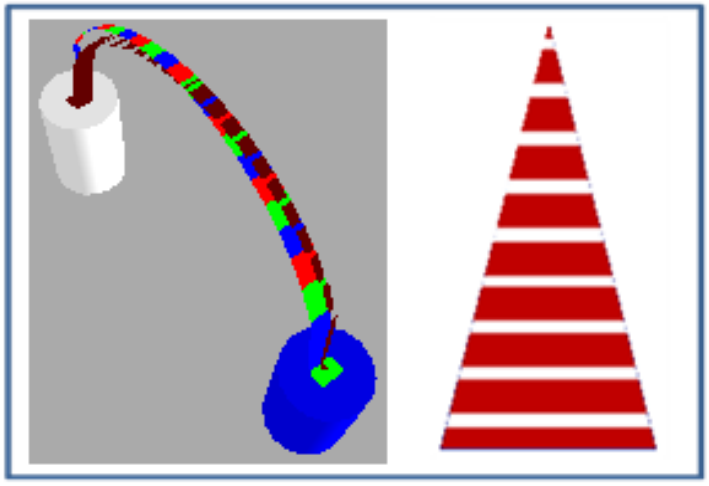


Figure 3.7 : Représentation des appels et des retours.

b) Algorithme de placement

Comme indiqué plus tôt, la vue globale permet de visualiser la trace combinée progressivement. Cette trace est élaguée pendant le processus de parcours. En effet, l'analyste peut enlever des messages et des objets qui sont inutiles dans le comportement abstrait que nous voulons capturer avec le diagramme de séquence. Le processus interactif est détaillé dans la section 4.4.3.

La trace combinée est exprimée sous forme d'un arbre. Pour placer les noeuds de cet arbre, c'est-à-dire les cylindres correspondant aux méthodes, nous plaçons d'abord le noeud racine au centre de la grille. La grille est alors divisée en quatre parties délimitées par des diagonales comme indiqué dans la Figure 3.8. Les noeuds fils de la racine sont assignés à chacune des quatre parties triangulaires.

Si le noeud racine a plus de quatre fils, les fils sont alors distribués radialement autour de la racine, laissant potentiellement des éléments de grille vide autour de la racine. Chaque fils devient alors la pointe d'une région triangulaire assignée au sous-arbre de son fils correspondant. Les fils sont donc récursivement affectés dans les régions triangulaires. Les sous-arbres d'une même région sont placés selon l'algorithme de positionnement de Walker [19].

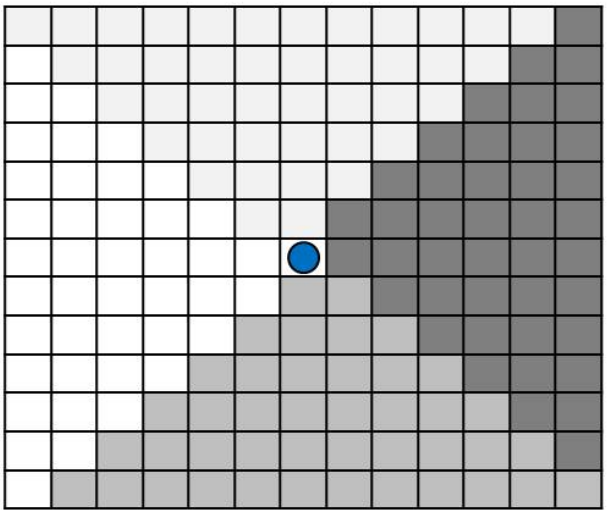


Figure 3.8 : Première étape de l'algorithme de placement.

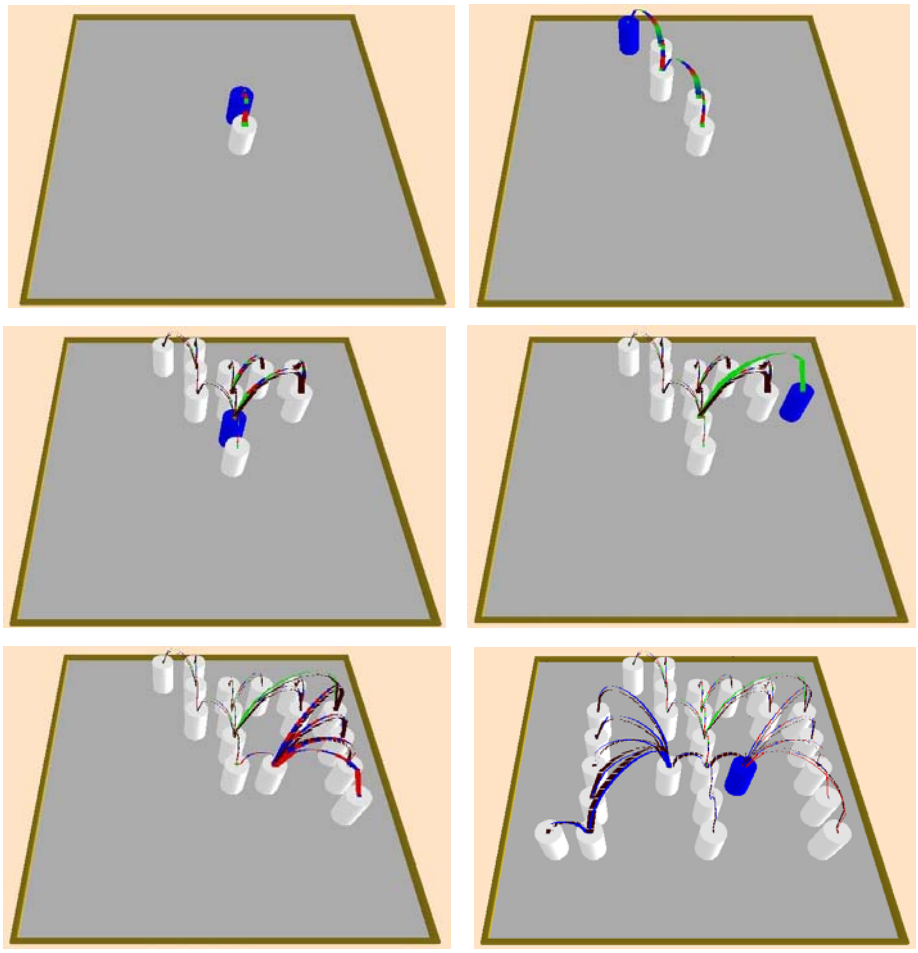


Figure 3.9 : Clichés de l'évolution de la vue globale.

- Algorithme de Walker

L'algorithme de Walker permet le positionnement récursif des noeuds d'un arbre en utilisant l'arborescence classique. Comme nous utilisons une grille dans notre cas, nous plaçons les noeuds d'un niveau dans la région triangulaire assignée au sous-arbre. Si la région ne contient pas assez de cellules, nous passons au niveau suivant jusqu'à ce que tous les noeuds soient placés. La taille de notre grille n'est pas fixée a priori et les niveaux peuvent être ajoutés si nous avons besoin de plus d'espace. La Figure 3.9 montre les clichés du positionnement pendant le parcours de la trace combinée.

L'algorithme de Walker est constitué de deux phases : dans la première phase, l'arbre est parcouru à partir des feuilles vers la racine (parcours postfixe) en calculant les placements préliminaires des sous-arbres indépendamment les uns des autres, autrement,

la racine de chaque sous-arbre sera centrée par rapport à ces fils. Dans la deuxième phase, l'arbre est parcouru dans le sens inverse (parcours préfixe) auquel nous calculons les placements réels de chaque nœud en assurant la symétrie de la forme de l'arbre.

3.4.2.2. Vue diagramme

Lors du parcours de la trace combinée, nous dessinons le diagramme de séquence selon les transformations automatisées de la section 3.4.1, ainsi que les actions exécutées par l'analyste.

Nous utilisons les notations standards d'UML2.0 [11] pour dessiner le diagramme de séquence. Pour aider l'analyste à prendre de meilleures décisions, nous ajoutons des informations sur les traces dans les messages extraits. Par conséquent, les flèches de messages sont élargies selon le nombre d'exécutions. Chaque flèche est colorée selon la trace. Les identificateurs des traces sont aussi ajoutés avant l'étiquette de message.
La Figure 3.10 montre une portion du diagramme de séquence obtenu de notre exemple. Le premier participant est ajouté pour indiquer l'acteur qui déclenche le scénario qui consiste à dessiner une forme géométrique. Les trois autres participants sont les objets créés pendant l'exécution. Ainsi, les quatre messages affichés existent dans les trois traces T1, T2 et T3.

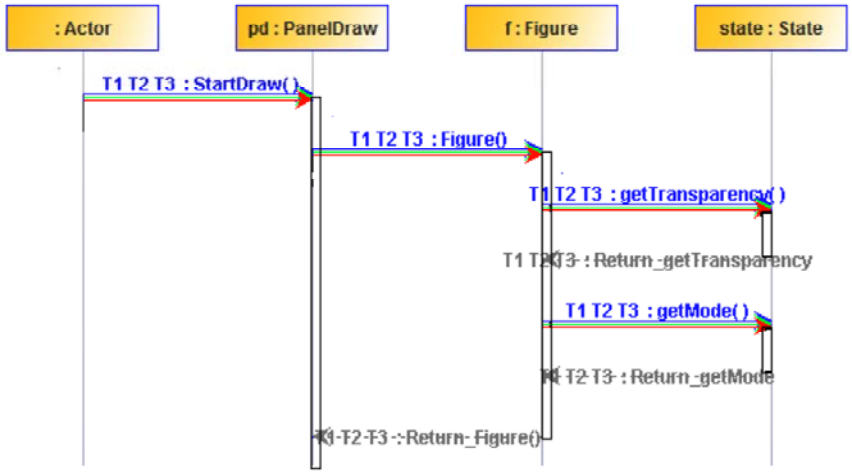


Figure 3.10 : Une portion de la vue diagramme.

3.4.4. Interactions

On pourrait interpréter l'extraction du diagramme de séquence comme un parcours progressif de la trace combinée. Dans les sous-sections suivantes, nous présentons comment l'analyste navigue dans la trace combinée et comment il modifie les traces.

3.4.3.1. Navigation dans la trace combinée

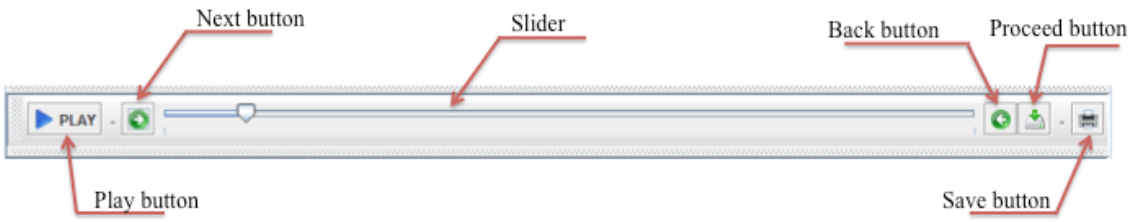


Figure 3.11 : Barre de navigation.

En mode par défaut, le parcours de la trace se fait en fonction des événements d'exécution. Comme la trace correspond à un arbre, le processus de parcours se réalise en profondeur de la racine vers les feuilles. Afin de permettre à l'analyste de naviguer, nous avons mis à sa disposition une barre de navigation avec une jauge de temps (Figure 3.11). Les différentes actions de navigation sont détaillées au tableau 3.1.

Tableau 3.1 : Commandes de la barre de navigation.

Commandes	Descriptions
Play	Permet d'initialiser et de lancer la navigation et la création du diagramme de séquence.
Next	Permet d'exposer l'appel de méthode suivant dans la vue globale et d'ajouter les éléments correspondants dans la vue diagramme.
Back	Permet d'annuler la dernière étape de navigation en supprimant les éléments des deux vues relatifs à l'appel correspondant.
Proceed	Permet de passer au mode automatique en transformant toutes les autres parties de la trace, sans donner le contrôle à l'analyste.

Save	Permet d'enregistrer le diagramme de séquence obtenu.
Slider	Une composante graphique permettant la visualisation du niveau d'avancement par rapport au parcours de la trace combinée.

Suite à chaque action de navigation, et après avoir regardé la vue globale et les éléments ajoutés dans le diagramme de séquence pour le dernier appel de méthode visité, l'analyste prend le contrôle où il veut modifier le diagramme de séquence et la trace combinée. Les modifications peuvent être un renommage des objets et des messages, une suppression des objets et des messages, ou une fusion des fragments du diagramme.

3.4.3.2. Renommage des objets et des messages

Les objets participant dans le diagramme de séquence sont identifiés par leurs adresses mémoire. Quand le diagramme de séquence est utilisé dans le but de documentation, le nom devrait aider à mieux comprendre le comportement. C'est particulièrement vrai quand plus qu'un objet est créé à partir de la même classe. Dans ce contexte, l'analyste peut renommer des objets avec des noms plus significatifs selon la documentation des cas d'utilisation. L'analyste pourrait aussi inspecter le code correspondant à la méthode appelante. Finalement, pour aider l'analyste à découvrir des noms, notre environnement propose aussi une liste de noms tirés par l'analyse du code correspondant (les variations des noms de variables où l'objet est stocké après sa création). Par exemple, l'acteur initialement créé avec l'étiquette « 1035198 : Circle » pourrait être renommé « circle : Circle ».

De même, les noms de messages peuvent être renommés par l'analyste. Dans notre exemple d'outil de dessin, une méthode nommée « m_cc () » modifie la couleur dans la classe « Circle ». Le message correspondant à l'appel « m_cc () » pourrait être renommé par « modify_object_color () » par exemple pour faciliter la compréhension du diagramme de séquence. En renommant des messages, les équivalences de noms sont stockées pour faciliter des tâches futures de maintenance.

3.4.3.3. Suppression des objets et des messages

Les traces d'exécution contiennent beaucoup de détails d'implémentation qui ne devraient pas apparaître dans un diagramme de séquence. Des exemples de ces détails sont des objets et des méthodes d'une bibliothèque, d'une interface graphique, temporaires, etc. La présence de ces détails est une des limitations majeures indiquées dans l'état de l'art (voir [3] par exemple). En regardant des noms de messages et d'objets, et en inspectant le code, l'analyste pourrait décider qu'un objet ou un message n'est pas utile pour le niveau abstrait ciblé.

Par défaut, en enlevant un objet de la trace combinée, tous les messages entrants et sortants associés sont éliminés. Les messages qui résultent de ces messages sont aussi récursivement enlevés. Dans beaucoup de cas, cette suppression totale n'est pas souhaitable. En effet, pour des raisons d'implémentation un objet à supprimer pourrait être utilisé comme un objet intermédiaire. Ainsi, nous ne voulons pas perdre le reste de l'interaction après l'action de suppression. Pour éviter ces situations, nous avons présenté trois opérations de suppression d'objet : (1) suppression récursive par défaut, (2) suppression avec la redirection de message (par exemple, lors de la suppression d'un objet *b*, si un message *m1* d'un objet *a* vers *b* déclenche un message *m2* de *b* vers un objet *c*, alors nous pouvons choisir d'ajouter *m1* de *a* vers *c* ou ajouter *m2* de *a* vers *c*), et (3) application de l'une des deux précédentes « suppressions », mais à tous les objets dans la trace ayant le même type que l'objet enlevé.

Comme pour les objets, un message pourrait aussi être retiré en appliquant les trois situations de la suppression des objets: (1) suppression récursive par défaut (tous les messages déclenchés par le message supprimé sont supprimés d'une façon récursive), (2) suppression avec redirection (par exemple, si nous avons la chaîne *m1*> *m2*> *m3*, nous pouvons supprimer *m2* et ajouter la chaîne *m1*> *m3*), et (3) application de l'une des deux précédentes « suppressions », mais à tous les messages correspondants à la même méthode dans la trace.

3.4.3.4. Fusion des fragments

Pendant la combinaison des traces d'exécution (section 3.3.2), nous alignons aussi des sous-arbres sur des appels de méthode communs et dont les objets ne sont pas du même type, mais ayant une super classe commune (les constructeurs sont aussi alignés). Dans la trace combinée, nous n'avons pas fusionné les sous-arbres alignés, mais nous injectons une action de recommandation. Cette action est déclenchée quand tous les sous-arbres alignés sont parcourus et après la construction de leurs éléments dans le diagramme de séquence. L'analyste pourrait visualiser les vues globale et de diagramme (et probablement le code) et accepter/rejeter la recommandation. En cas d'acceptation, les ordres de messages associés sont fusionnés dans le diagramme de séquence et les objets sont remplacés par un objet au type de l'ancêtre commun.

Dans notre exemple, les trois traces sont consacrées à la création et au dessin respectivement d'un cercle, d'un rectangle et d'un segment. Les événements 9 et 10 dans la Figure 3.2 et les événements 8 et 9 dans la Figure 3.3 sont les appels de méthodes correspondant à ce comportement commun. Dans ces deux cas (plus le cas du segment), le constructeur est appelé, suivi par la méthode « `draw`». Nous présentons la partie du diagramme de séquence qui correspond à ces parties dans la Figure 3.12 (a). Après la fusion, les trois objets «`c:Circle`», «`r:Rectangle`» et «`seg:Segment`» sont remplacés par l'objet «`g:GraphicRepresentation`» parce que «`Circle`», « `Rectangle`» et « `Segment`» ont une super classe «`GraphicRepresentation`» commune. Les constructeurs sont remplacés par le constructeur du nouvel objet. Finalement, le message «`draw()`» avec les trois flèches colorées entre le nouvel objet et « `f:Figure`» remplace les trois messages « `draw` () » (Figure 3.12 (b)).

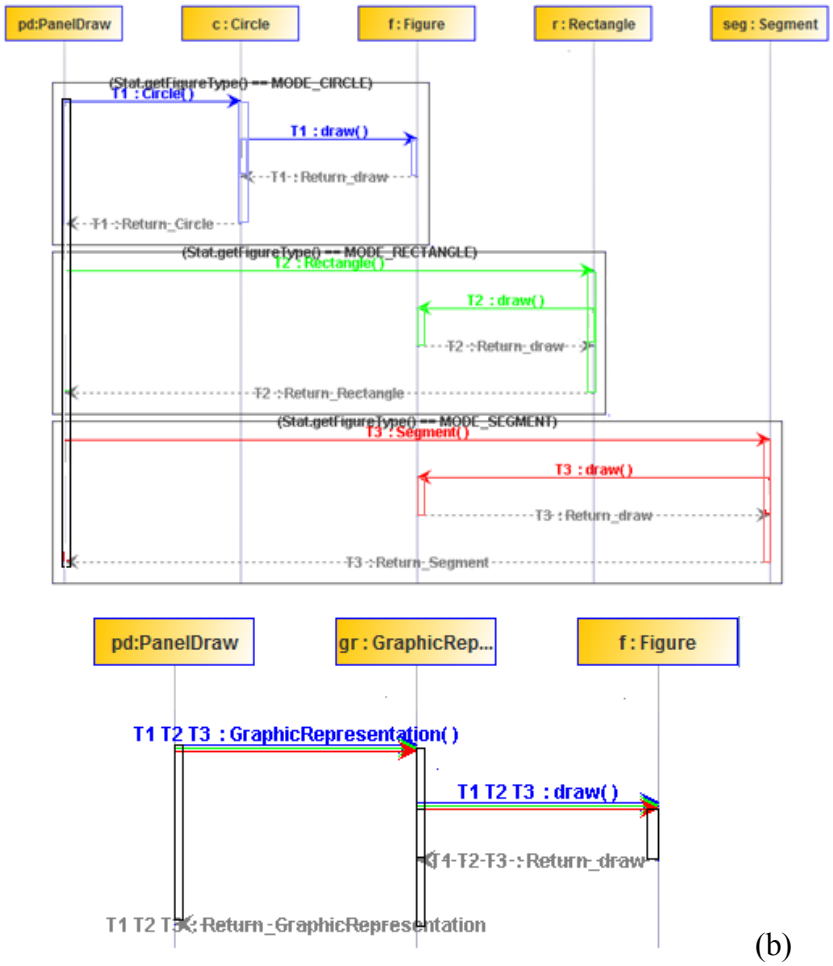


Figure 3.12 : Fusion des fragments de diagramme de séquence.

3.5. Conclusion

À travers ce chapitre, nous avons présenté notre approche pour la rétro-ingénierie de diagrammes de séquence qui touche à différents aspects du génie logiciel. Nous distinguons trois étapes : D'abord, on génère un ensemble de traces d'exécution correspondant à un scénario spécifique par l'instrumentation du code. Ensuite, on aligne ces traces en se basant sur l'algorithme de Smith et Waterman [16]. Enfin, on extrait le diagramme de séquence en utilisant notre système de visualisation interactif. Cette interactivité offre une flexibilité de navigation, d'ajout, de suppression et de fusion lors de la construction des diagrammes de séquence qui servent à la compréhension des

systèmes de grande taille. Dans le chapitre suivant, nous présentons deux études de cas pour montrer la faisabilité de notre approche.

Chapitre 4

Étude de cas

4.1. Introduction

Notre approche de rétro-ingénierie de diagrammes de séquence est semi-automatique : l'utilisateur doit prendre des décisions à certains moments lors du processus de l'extraction du diagramme de séquence. Notre système permet de guider l'utilisateur à produire un diagramme de séquence simple et facile à comprendre. L'objectif de cette étude de cas est d'illustrer comment la visualisation interactive complète la rétro-ingénierie automatisée des diagrammes de séquence et permet d'accomplir que ces diagrammes soient fidèles au système étudié.

Dans le reste de ce chapitre, nous détaillons le déroulement et les résultats de notre étude de cas réalisée pour valider notre travail.

4.2. Mise en place

4.2.1. Système ATM

Pour effectuer notre étude, nous avons choisi un système de simulation d'un ATM[7]. Ce système Java contient 24 classes. Une autre implémentation en C++ d'un même système a été utilisée pour valider l'approche automatisée présentée par Briand et al. [3]. Outre le fait que ce système ait été utilisé précédemment pour le même problème, le système de simulation d'un ATM a l'avantage que les documents d'analyse et de conception sont disponibles. Ces documents comprennent les cas d'utilisations qui sont nécessaires pour déterminer les scénarios des diagrammes de séquence à extraire. Ils comprennent également les diagrammes de séquence correspondants à différents

[7] http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/

scénarios. Cela nous permet de comparer nos résultats d'une manière objective. Nous avons sélectionné trois scénarios de cas d'utilisations : «`Session`», «`Deposit`» et «`Withdraw`» pour chaque scénario.

4.2.2. Sujets

Cette étude de cas a été menée sur cinq sujets volontaires, des étudiants gradués de notre département informatique et de recherche opérationnelle. Ces sujets avaient de bonnes connaissances des modélisations et conceptions UML et en particulier des diagrammes de séquence. Nous avons demandé à ces volontaires d'expérimenter avec notre environnement pour extraire les diagrammes de séquence des trois scénarios sélectionnés. Durant le processus de génération, nos sujets ont eu accès seulement au code et au Javadoc du système de simulation d'un ATM. L'expérience a eu lieu sur le même ordinateur et le même écran. Tous nos sujets ont reçu une courte formation d'environs 20 minutes pour se familiariser avec les concepts et les contrôles de notre outil.

4.2.3. Principes d'analyses

Afin de valider de notre approche, nous nous sommes intéressés dans un premier temps aux résultats individuels de chaque sujet. En effet, nous avons procédé à la comparaison des trois diagrammes de séquence pour chaque scénario : le premier diagramme est le diagramme de conception (DD : Design Diagram) fourni avec la documentation du système ATM; il représente notre référence pour la comparaison. Le second diagramme (ATD : Automated Transformation Diagram) est produit en utilisant une transformation automatisée selon les règles de transformation de Briand et al. [3]. Le troisième diagramme (IVD : Interactive Visualization Diagram) est produit par un sujet en utilisant notre environnement de visualisation interactif. Pour chaque diagramme, nous avons compté le nombre de messages et le nombre d'acteurs. Pour ATD et IVD, nous avons compté le nombre de messages (respectivement acteurs) qui existent dans le diagramme de conception (DD).

Dans un second temps, nous avons étudié les résultats globaux pour tous les sujets, ainsi que les variations entre les diagrammes obtenus par les différents sujets en termes de

précision et de rappel. Nous calculons pour chaque diagramme obtenu les quatre variables suivantes:

$Précision_M = N_{MEdd} / NbT_M$ = nombre de messages existant dans DD / nombre total de messages.

$Rappel_M = N_{MEdd} / N_M^{DD}$ = nombre de messages existant dans DD / nombre total de messages de DD.

$Précision_A = N_{AEdd} / NbT_A$ = nombre d'acteurs existant dans DD / nombre total d'acteurs.

$Rappel_A = N_{AEdd} / N_A^{DD}$ = nombre d'acteurs existant dans DD / nombre total d'acteurs de DD.

Ensuite, nous calculons la moyenne de chacune de ces variables pour les diagrammes en utilisant notre approche.

Dans le reste de ce chapitre, nous présentons tout d'abord les résultats quantitatifs pour les trois scénarios pour un seul sujet. Ensuite, nous étudions les variations des résultats de nos sujets en termes de précision et de rappel. Enfin, nous détaillons les résultats du scénario « `Session` ».

4.3. Résultats et discussions

4.3.1. Résultats spécifiques à un seul sujet

Le tableau 4.1 résume le contenu du diagramme de conception et les diagrammes générés d'un seul sujet. ATD contient environ trois fois le nombre de messages de DD et entre deux et six acteurs supplémentaires, cependant le rappel des messages et des acteurs est de 100%. C'est parce que toutes les interactions qui se produisent dans l'exécution sont présentées dans ATD.

IVD contient beaucoup moins de messages non pertinents (réduction entre 30% et 60%). Le rappel est cependant moindre que l'approche automatisée. En effet, deux messages sont manquants dans les diagrammes correspondant aux scénarios «`Session`» et «`Withdraw`». Dans les deux cas, les messages manquants ont été enlevés parce que le sujet ne les considère pas importants pour la documentation du scénario.

Tous les acteurs du DD sont trouvés dans IVD à l'exception d'un seul dans les deux scénarios « `Session` » et «`Withdraw` ». Dans les deux cas, l'objet enlevé

«:Display» correspond à l'acteur « :ConsumerConsole» dans le diagramme de conception. Cet objet a été éliminé parce qu'il était considéré comme un objet de l'interface graphique. Étonnamment, cet objet n'a pas été supprimé dans le scénario « Deposit ». Un acteur a été ajouté aux scénarios « Deposit » et « Withdraw ». Il correspond à l'objet « :Session ». En DD, cet acteur n'a pas été inclus parce qu'il y a déjà un acteur qui représente la transaction « Withdraw » (respectivement « Deposit »).

Ces résultats devraient être contrastés par des considérations de performance. Pour tous les scénarios, le processus d'extraction pour chacun des diagrammes de séquence a pris moins de 20 minutes, incluant la consultation du code et du Javadoc. Ce temps d'exécution est en grande partie acceptable pour des tâches de rétro-ingénierie et pourrait être facilement justifié par l'amélioration de la précision des diagrammes produits par notre approche par rapport à ceux produits automatiquement.

Tableau 4.1 : Résultats quantitatifs du diagramme de conception et des diagrammes de séquence générés par le sujet 1.

Scénarios			Session	Deposit	Withdraw
Diagrammes de conception	Nb messages		10	13	15
	Nb acteurs		5	6	6
Transformation automatique	Nb messages	Tous	39	39	41
		Pertinents	10	13	15
	Nb acteurs	Tous	11	8	8
		Pertinents	5	6	6
Visualisation interactive	Nb messages	Tous	16	27	29
		Pertinents	8	13	13
	Nb acteurs	Tous	4	7	6
		Pertinents	4	6	5

4.3.2. Résultats globaux pour tous les sujets

Nous nous intéressons d'abord à comparer les diagrammes de séquence obtenus par notre approche et les diagrammes obtenus suite à des transformations automatiques comme celle de Briand et al. [3]. Pour cela, nous calculons la moyenne des variables de précision et de rappel pour nos cinq sujets pour chaque scénario et pour chacun des acteurs et des messages.

Les histogrammes présentés dans les figures (
Figure 4.1 et Figure 4.2) résument nos résultats obtenus. Nous exposons pour chaque scénario sous forme d'histogrammes la précision et le rappel des diagrammes générés suite à des transformations automatiques (AT) et la moyenne des précisions et des rappels concernant les diagrammes obtenus suite à l'application de notre approche par nos sujets (MIV).
La
Figure 4.1 illustre une augmentation de la précision des messages dans chacun des scénarios, soient de 26% à 48% dans le scénario « Session », de 33% à 52% dans le scénario «Deposit» et de 37% à 48% dans le scénario «Withdraw». Cependant contrairement à la précision, nous constatons une diminution du rappel de 100% à 89% dans le scénario «Session», de 100% à 95% dans le scénario « Deposit » et de 100% à 93% dans le scénario «Withdraw».
L'analyse de la Figure 4.1 montre que la mesure précision augmente dans le cas des trois scénarios, ce qui est évident suite à l'utilisation de notre environnement interactif et par conséquent, de l'élimination de plusieurs messages considérés inutiles afin de produire un diagramme de séquence compréhensif et qui englobe toutes les situations possibles. Cette augmentation est plus prononcée dans le scénario «Session», ce qui est aussi logique puisque juste dans ce scénario « Session », on illustre un cas de fusion de fragments de diagramme de séquence recommandé par notre système (ceci sera détaillé dans la section suivante). Toutefois, les mesures de rappels diminuent faiblement dans les trois scénarios. Cette diminution est causée par l'élimination de quelques messages considérés importants en référence par rapport au diagramme de conception.

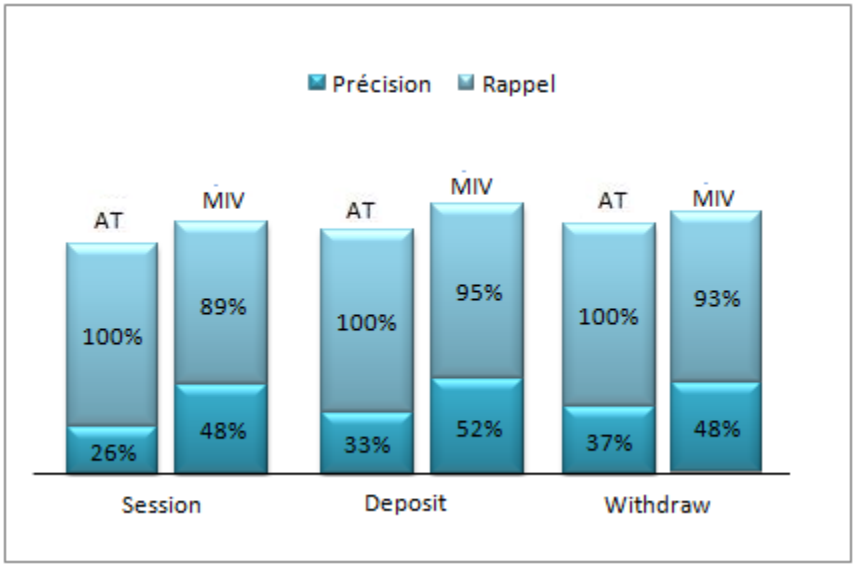


Figure 4.1 : Précision et rappel des **messages** entre les diagrammes obtenus par les transformations automatiques (AT) et la moyenne de ceux obtenus par notre approche (MIV) pour chacun des trois scénarios «Session», «Deposit» et «Withdraw».

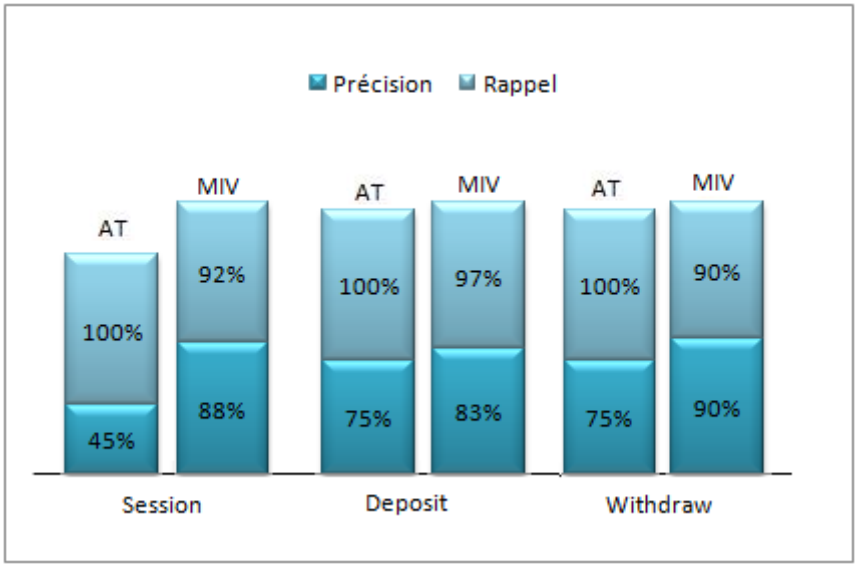


Figure 4.2 : Précision et rappel des **acteurs** entre les diagrammes obtenus par les transformations automatiques (AT) et la moyenne de ceux obtenus par notre approche (MIV) pour chacun des trois scénarios « Session », « Deposit » et « Withdraw ».

De même que la Figure 4. 1, la Figure 4. 2 illustre la variation de précision et de rappel des messages. Nous remarquons ainsi une augmentation de la précision des messages dans chacun des scénarios, soit de 45% à 88% dans le scénario «Session», de 75% à 83% dans le scénario «Deposit» et de 75% à 90% dans le scénario «Withdraw». Cependant contrairement à la précision, nous constatons une diminution du rappel de 100% à 92% dans le scénario «Session», de 100% à 97% dans le scénario « Deposit » et de 100% à 90% dans le scénario « Withdraw ».

Comme au niveau des messages, l'analyse de la Figure 4.2 au niveau des acteurs montre que les variations de la précision augmentent dans les trois scénarios. Cette augmentation est plus prononcée dans le scénario « Session ». Toutefois, les variations des rappels diminuent dans les trois scénarios et cette diminution est plus prononcée dans le scénario « Session ».

La comparaison des figures 4.1 et 4.2 (entre les mesures au niveau des messages et des acteurs) montre que les deux mesures suivent la même variation, toutefois, l'amplitude de la variation est plus importante dans le cas de la mesure précision au niveau des acteurs (Figure 4.2). Cette variation est causée par le fait que dans le diagramme de séquence, le nombre d'acteurs est toujours moindre que le nombre de messages, et par conséquent, il est plus facile de faire le choix entre garder ou éliminer des acteurs que des messages.

Afin de valider l'efficacité de notre outil de rétro-ingénierie des diagrammes de séquence, nous nous intéressons maintenant à la variabilité entre nos sujets de l'expérience et par conséquent aux diagrammes de séquence extraits. Pour cela, nous employons les mêmes mesures (précision et rappel) calculées précédemment.

Nous exposons dans la Figure 4.3 et la Figure 4.4 la variation des variables de précision et de rappel des diagrammes générés par nos sujets pour le scénario « Session », respectivement au niveau des messages et au niveau des acteurs.

Au niveau des messages, l'analyse de la Figure 4.3 montre qu'une augmentation de la précision entraîne une diminution de celle du rappel. Ainsi, nous concluons que la relation entre les deux mesures est inverse. Par exemple dans le cas du sujet 4 nous remarquons que le niveau du rappel est de 100% alors que celui de la précision est

d'environ 40%. Par contre dans le cas du sujet 2, le niveau de précision est plus prononcé (environ 60%) alors que celui du rappel est plus de 80%.

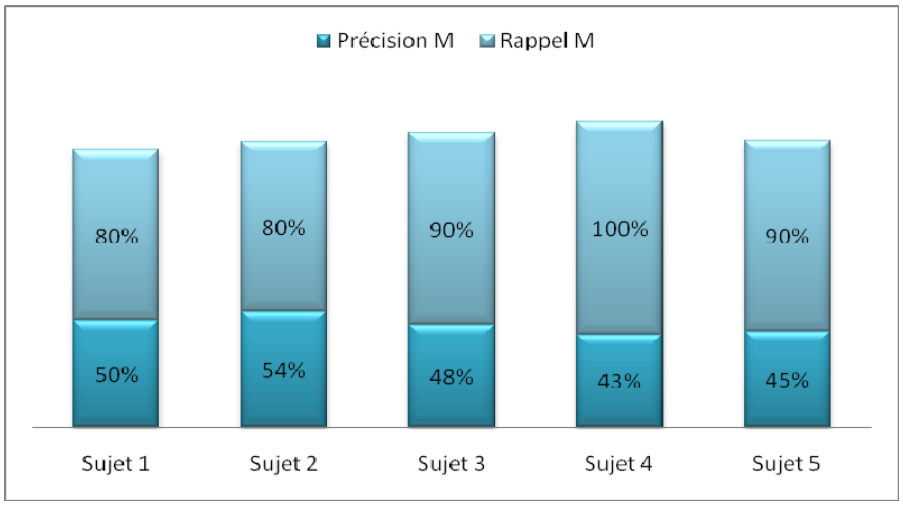


Figure 4.3 : Précision et rappel des **messages** entre les diagrammes générés par les différents sujets pour le scénario «Session».

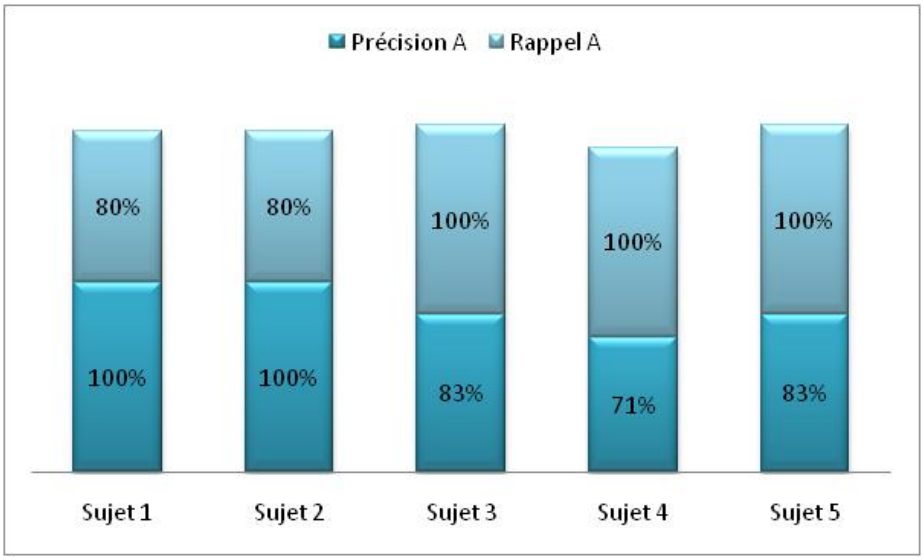


Figure 4.4 : Précision et rappel des **acteurs** entre les diagrammes générés par les différents sujets pour le scénario «Session».

Nous concluons de ces résultats que l'efficacité de l'extraction des diagrammes de séquence varie faiblement selon les compétences de l'utilisateur, dû au fait que la majorité des messages sont des messages de retour indiquant l'achèvement d'une activité et des messages d'initialisation indiquant la création de nouveaux objets. Ceci engendre des variations de choix entre nos sujets (soit de les garder ou de les éliminer). Par conséquent, les diagrammes résultants ne sont pas totalement identiques.

Nous remarquons aussi que la précision des messages ne dépasse pas 60%, dû au fait qu'une portion des messages de détails (retour, construction, …) n'ont pas été supprimés. Ces messages ont été considérés importants par les sujets, même s'ils n'apparaissent pas dans le diagramme de conception.

Contrairement à la Figure 4.3, au niveau des acteurs, la Figure 4.4 montre que la précision des acteurs est élevée (100% pour les sujets 3, 4 et 5), ce qui indique que la majorité (ou la totalité) des objets gardés dans les diagrammes de séquence sont pertinents (présents dans le diagramme de conception).

Nous remarquons que dans le cas des deux premiers sujets, la précision est plus importante en pourcentage que le rappel. La position s'inverse dans le cas des sujets 3, 4 et 5. Par exemple dans le cas du sujet 4, le rappel est de 100% alors que celui de la précision est d'environ 70%, cependant, dans le cas du sujet 2 la précision est plus importante (100%) alors que le rappel est de 80%. Par la suite et conformément à la Figure 4.3, la Figure 4.4 illustre une relation inverse entre les deux mesures.

On s'aperçoit aussi que les diagrammes de séquence obtenus par application directe de notre approche ont conservé la sémantique des données, tout en simplifiant encore plus les diagrammes de séquence préalablement obtenus par l'application des transformations automatiques.

En ce qui concerne le temps d'exécution, on a constaté d'après les expérimentations réalisées que les temps consacrés pour générer les diagrammes de séquence sont raisonnables. Néanmoins, l'utilisateur ne parvient pas parfois à comprendre immédiatement les fonctionnalités du système étudié, c'est-à-dire, il doit y avoir une période de temps pour qu'il puisse s'adapter à eux. Par exemple, parfois l'usager ne peut

décider de supprimer ou de garder une certaine entité que lorsqu'il progresse dans la trace combinée, alors qu'il serait plus adéquat de pouvoir le faire instantanément.

4.3.3. Scénario « Session »

Pour mieux illustrer la contribution de la visualisation interactive, nous détaillons le scénario «Session». Ce scénario est nécessaire à l'exécution d'autres scénarios plus spécifiques pouvant être effectués à l'aide du système ATM. Pour produire assez de variations pour ce scénario, nous avons exécuté le système avec trois opérations différentes: « `Withdraw` », « `Deposit`» et « `Transfer` ». Ce scénario pourrait être décrit comme suit. Tout d'abord, l'utilisateur est invité à insérer la carte et à composer son NIP. La carte et le NIP sont ensuite vérifiés. Une fois la validation effectuée, l'utilisateur lance l'exécution d'une transaction spécifique.

Le schéma de conception DD de ce scénario est présenté à la Figure 4.5. Le diagramme généré avec notre approche interactive IVD par le sujet 1 est présenté dans la Figure 4.6 et le diagramme généré suite aux transformations automatiques ATD est représenté dans la Figure 4.7. La première observation est que de nombreux objets participants sont présents dans ATD en raison de la phase d'initialisation qui n'existe pas dans DD. Les messages correspondant à cette phase, tels que « `Bank` () » et « `CardReader` () », ont été enlevés par notre sujet 1 dans IVD, car il les considère comme des détails d'implémentation. Cela entraîne la suppression des acteurs correspondants (par exemple: « :`Bank` » et «:`CardReader` »).

La deuxième observation est que deux acteurs considérés comme objets d'interface ont été enlevés par notre sujet 1. Dans le cas de « :`keyboard` », ceci est considéré comme une bonne décision (absent dans DD). Toutefois, dans le cas de «:`Display`», le concepteur estime que cet acteur (appelé «:`CustomerConsole`») est important pour la documentation. Dans les deux cas, la suppression de ces acteurs a abouti à la suppression des messages correspondants tels que « `requestCard` () » entre « :`ATM` » et « :`Display` », et «`readPin` ()» entre «: `ATM`» et «: `KeyBoard`».

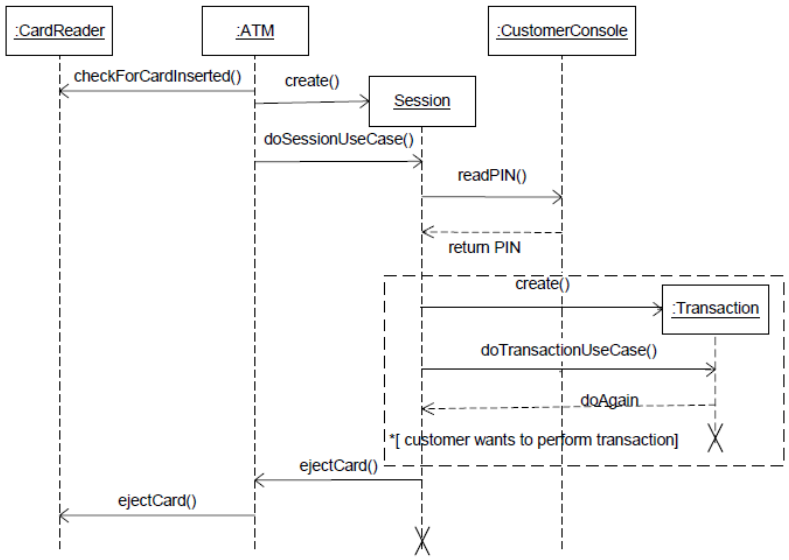


Figure 4.5 : Diagramme de séquence de conception pour le scénario « *Session* ».

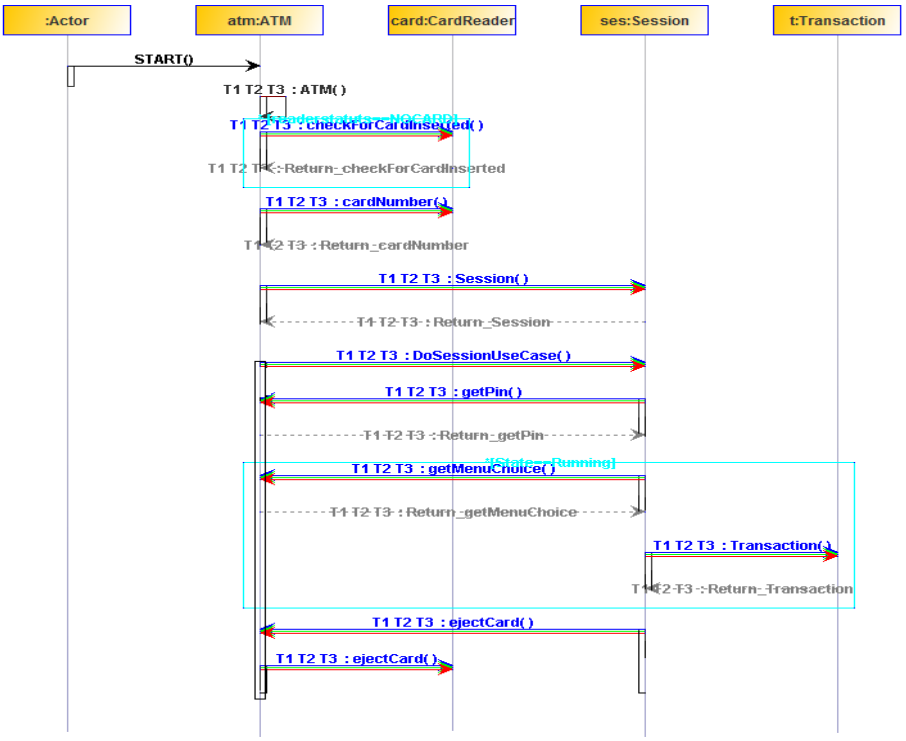


Figure 4.6 : Diagramme de séquence pour le scénario « *Session* » généré par la visualisation interactive.

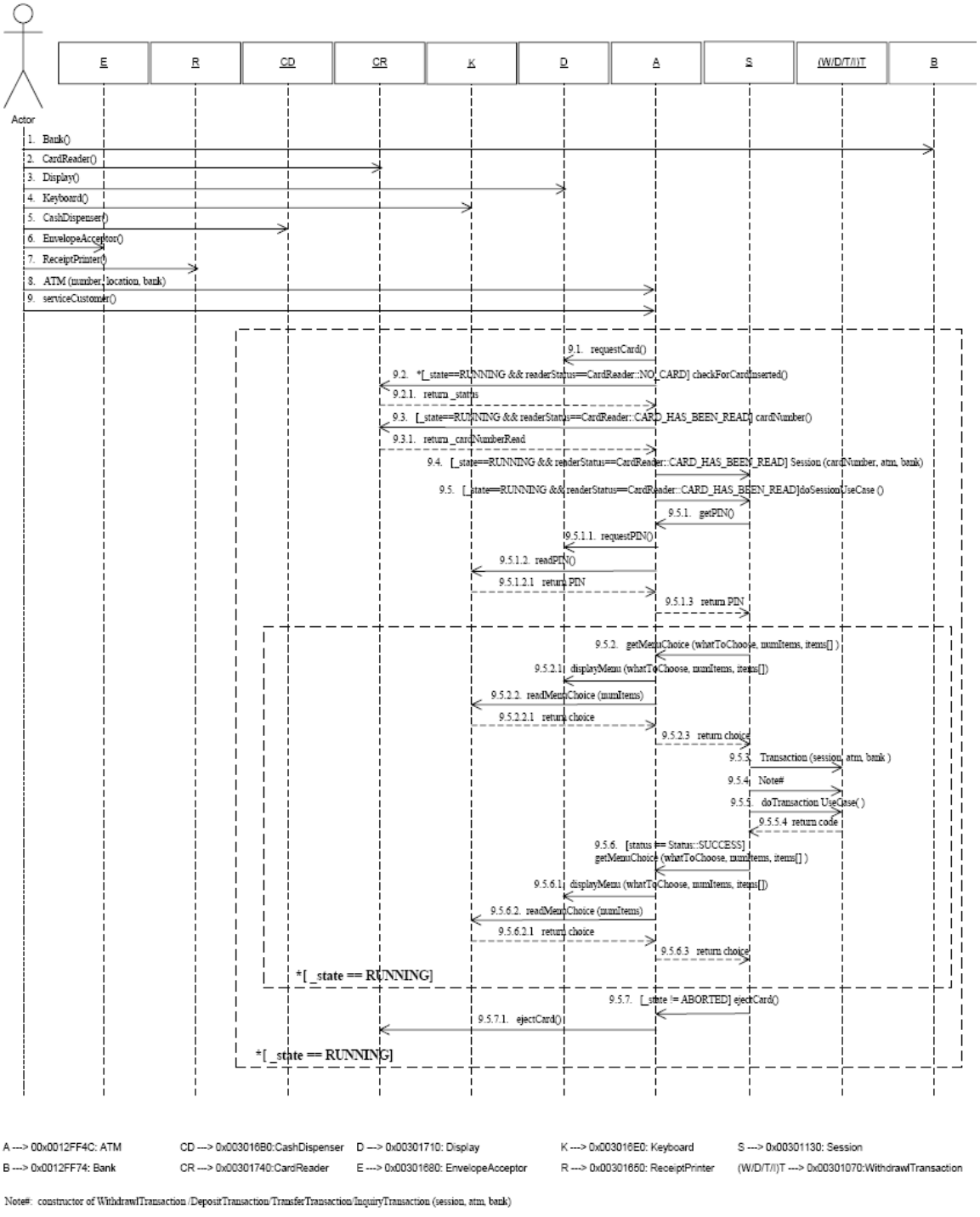


Figure 4.7 : Diagramme de séquence pour le scénario « *Session* » généré par les transformations automatiques de Briand et al. [3].

Une autre observation intéressante concerne la fusion des fragments d'exécution dans IVD. Les messages appartenant aux transactions spécifiques (deux par transaction) sont remplacés par deux messages. Les acteurs spécifiques sont également remplacés par « :`Transaction` », qui correspond à un objet de la classe abstraite « `Transaction` » qui est la super classe commune des classes «`WithdrawTransaction`», «`DepositTransaction` » et «`TransferTransaction` ». Dans ATD (la Figure 4.7), il existe un seul acteur étiqueté par les quatre transactions. Toutefois, à notre avis, il s'agit d'un acteur spécifique pour chaque exécution et la fusion entre les quatre traces correspondantes ne se fait pas automatiquement.

Suite à ces analyses, on peut déduire que notre approche est vraiment efficace puisqu'elle donne de meilleurs résultats que ceux obtenus par les outils automatiques de rétro-ingénierie.

4.4. Conclusion

La visualisation interactive combinée avec des capacités de recommandation permet de réduire la complexité des traces d'exécution. La principale différence des autres approches avec notre travail est l'interactivité. En effet, nous utilisons la visualisation non seulement pour afficher les résultats d'un processus, mais aussi pour permettre à un analyste de contribuer au processus.

Dans ce chapitre, nous avons présenté les résultats de nos études de cas sur l'efficacité de notre approche de rétro-ingénierie des diagrammes de séquence, basée sur des techniques de visualisation interactives. Nous terminons notre travail, présenté dans le cadre de notre projet de maîtrise, par une conclusion et des perspectives futures.

Chapitre 5

Conclusion

La rétro-ingénierie de diagrammes de séquence est une tâche importante pour la compréhension du comportement des programmes orientés objet. Entre autres, la rétro-ingénierie a comme but la redocumentation, la maintenance et l'évolution des anciennes applications. En effet, ce processus est d'autant plus complexe que l'application est mal structurée, trop ancienne, ou non ou mal documentée. Plusieurs outils ont été proposés pour automatiser cette tâche. Par contre, et comme démontré dans la partie sur l'état de l'art, ces outils n'offrent pas une automatisation complète en considération de la complexité du problème.

Ce mémoire propose une approche semi-automatique basée sur l'analyse dynamique et la visualisation interactive. Notre approche suit trois étapes. D'abord, les traces d'exécution sont produites à partir d'un scénario précis. Ces traces d'exécution sont ensuite alignées pour produire une trace combinée qui englobe le comportement général du même scénario. La troisième étape est consacrée à l'extraction du diagramme de séquence. L'extraction combine les transformations automatiques au diagramme de séquence, avec des actions de l'analyste en utilisant notre environnement de visualisation interactif. Notre environnement inclut un module de recommandation qui suggère des possibilités d'abstractions pour le comportement du système étudié.

Nous avons évalué notre approche par une étude de cas, sur le même système que celui étudié par Briand et al. [3] (Système de simulation ATM). Notre étude de cas a été menée avec cinq sujets. D'abord, nous avons analysé les résultats individuels des sujets (les diagrammes produits en utilisant notre approche) par rapport aux diagrammes de conception et aux diagrammes produits automatiquement. Ensuite, nous avons étudié la variation entre les résultats de nos sujets en termes de précision et de rappel. Nos résultats ont montré que la visualisation interactive permet de produire des diagrammes plus concis avec un comportement plus abstrait. Le temps consacré à l'interaction est

acceptable compte tenu de la nature de la tâche de rétro-ingénierie et des limitations des approches automatiques.

La structure de ce mémoire reflète notre démarche pour résoudre le problème de génération des diagrammes de séquence. Nous avons effectué un survol des différents travaux en relation de notre domaine de recherche (analyse dynamique du code, visualisation du logiciel, techniques interactives utilisées dans le domaine de génie logiciel). Ensuite, nous avons détaillé les principes de notre approche. Enfin, nous avons présenté les résultats menés par nos études de cas.

Bien que les premiers résultats sont très prometteurs, il reste encore beaucoup à faire. Du point de vue de la visualisation, la vue globale pourrait grandement bénéficier de la perception des événements d'exécution. Du point de vue de l'interaction, le module de recommandation pourrait aussi envisager des cas plus subtils. En effet, nous prévoyons explorer l'apprentissage incrémental et progressif avec la rétroaction de l'analyste. D'autres travaux futurs incluent aussi la rétro-ingénierie d'autres diagrammes dynamiques tels que les diagrammes d'état. Nous pensons que la visualisation interactive pourrait aider la définition des états abstraits qui sont difficiles à déterminer automatiquement.

BIBLIOGRAPHIE

[1] C. Ghezzi, M. Jazayeri, et D. Mandrioli, “Fundamentals of Software Engineering” Prentice Hall International Ed., 1991.

[2] A. Rountev, O. Volgin, et M. Reddoch, “Static Control Flow Analysis for Reverse Engineering of UML Sequence Diagrams”, Workshop on Program analysis for software tools engineering, pp. 96-102, 2005.

[3] L. Briand, Y. Labiche, et Y. Miao, “Towards the Reverse Engineering of UML Sequence Diagrams”, Working Conference on Reverse Engineering, pp. 57-66, WCRE 2003.

[4] L. Briand, Y. Labiche, et Y. Miao, “Towards the Reverse Engineering of UML Sequence Diagrams for Distributed, Multithreaded Java Software”, IEEE Trans. on Software Engineering, pp. 642-643, 2006.

[5] K. Taniguchi, T. Ishio, T. Kamiya, S. Kusumoto, et K. Inoue “Extracting Sequence Diagram from Execution Trace of Java Program”, International Workshop on Principles of Software Evolution (IWPSE’2005), 2005, pp. 148-151.

[6] R. Delamare, B. Baudry, et Y. Le Traon, “Reverse Engineering of UML 2.0 Sequence Diagrams from Execution Traces”, Workshop on Object-Oriented Re-engineering, 2006.

[7] W. De Pauw, D. Kimelman, et J. Vlissides, “Modeling object-oriented program execution”, Proceedings of the 8th European Conference on Object-Oriented Programming, volume 821, pages 63-182. Springer-Verlag, July 1994.

[8] B. Cornelissen, A. Zaidman, et A. Deursen, “Trace Visualization for Program Comprehension: A Controlled Experiment”, In IEEE 17th International Conference on Program Comprehension, ICPC ’09. , pages 100–109, 2009.

[9] R. Kollman et M. Gogolla, “Capturing dynamic program behavior with UML collaboration diagrams”, In European Conference on Software Maintenance and Reengineering, pages 58-67, 2001.

[10] P. Tonella et A. Potrich, “Reverse engineering of the interaction diagrams from C++ code”, Conf. Software Maintenance, pages 159-168, 2003.

[11] R. Wettel et M. Lanza, “Visualizing software systems as cities”. 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007, , pages 92–99, 2007.

[12] S. Eick, J. L. Steffen, et E. Sumner, “Seesoft - a tool for visualizing line oriented software statistics”. IEEE Transactions on Software Engineering, 18(11):957–968, 1992.

[13] X. Zhou et K. Feiner, “Visual task characterization for automated visual discourse synthesis”, CHI 98: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 392–399, 1998. ACM Press/Addison-Wesley Publishing Co.

[14] B. Shneiderman, “The eyes have it: A task by data type taxonomy for information visualizations”. IEEE Visual Languages (UMCP-CSD CS-TR-3665), pages 336–343, 1996.

[15] W. De Pauw, D. Lorenz, J. Vlissides, et M. Wegman, “Execution Patterns in Object-Oriented Visualization”, Proc of the 4 th USENIX Conference on Object-Oriented Technologies and Systems (COOTS), 1998.

[16] T. F. Smith et M. S. Waterman, “Identification of common molecular subsequences”, Journal of Molecular Biology 147 (1981), 195-197.

[17] G. Langelier, “Visualisation de la qualité des logiciels de grandes tailles”, M.Sc. thesis, Université de Montréal, 2005.

[18] S. Hassaine, “Utilisation des textures pour la visualisation de logiciels de grande taille ”, M.Sc. thesis, Université de Montréal, 2007.

[19] J. Walker, “A node positioning algorithm for general trees”. Software Practice and Experience, 20 (7) : 685-705, 1990.

[20] G. Booch, J. Rumbaugh, et I. Jacobson. “The Unified Modeling Language User Guide”, Addison Wesley First Edition, October 20, 1998.

[21] B. Cornelissen, A. Zaidman, et A. Deursen, “A Controlled Experiment for Program Comprehension through Trace Visualization”, IEEE Trans. on Software Engineering, 2010.

[22] Robert K. Yin, “Case Study Research. Design and Methods” Second edition, Thousand Oaks 2008.

[23] A. Zaidman et S. Demeyer, “Automatic identification of key classes in a software system using webmining techniques”, Journal of Software Maintenance and Evolution, vol. 20, no. 6, pp. 387–417, 2008.

[24] H. Grati, H. Sahraoui, et P. Poulin, “Extracting Sequence Diagrams from Execution Traces using Interactive Visualization”, Working Conference on Reverse Engineering 2010.

[25] A. Zaidman et S. Demeyer, “Automatic identification of key classes in a software system using webmining techniques”, Journal of Software Maintenance and Evolution, vol. 20, no. 6, pp. 387–417, 2008.

[26] I. Philippow, D. Streitferdt, M. Riebisch, et S. Naumann. “An approach for reverse engineering of design patterns”, Software Systems Modeling, pages 55–70, 2005.

[27] J. Smith et D. Stotts. "SPQR: flexible automated design pattern extraction from source code”, In ASE ’03: The 18th International Conference on Automated Software Engineering, pages 215–224, 2003.

[28] J. Lim, T. Reps, et B. Liblit. “Extracting file formats from executables”, Proceedings of the 13th Working Conference on Reverse Engineering, Benevento, Italy, Oct. 23–27 2006.

[29] R. Wang, X. Wang, K. Zhang, et Z. Li"Towards automatic reverse engineering of software security configurations", Conference on Computer and Communications Security, Pages: 245-256, 2008.

ANNEXE : Diagrammes de séquences générés par les sujets pour le cas d'utilisation *Session*.

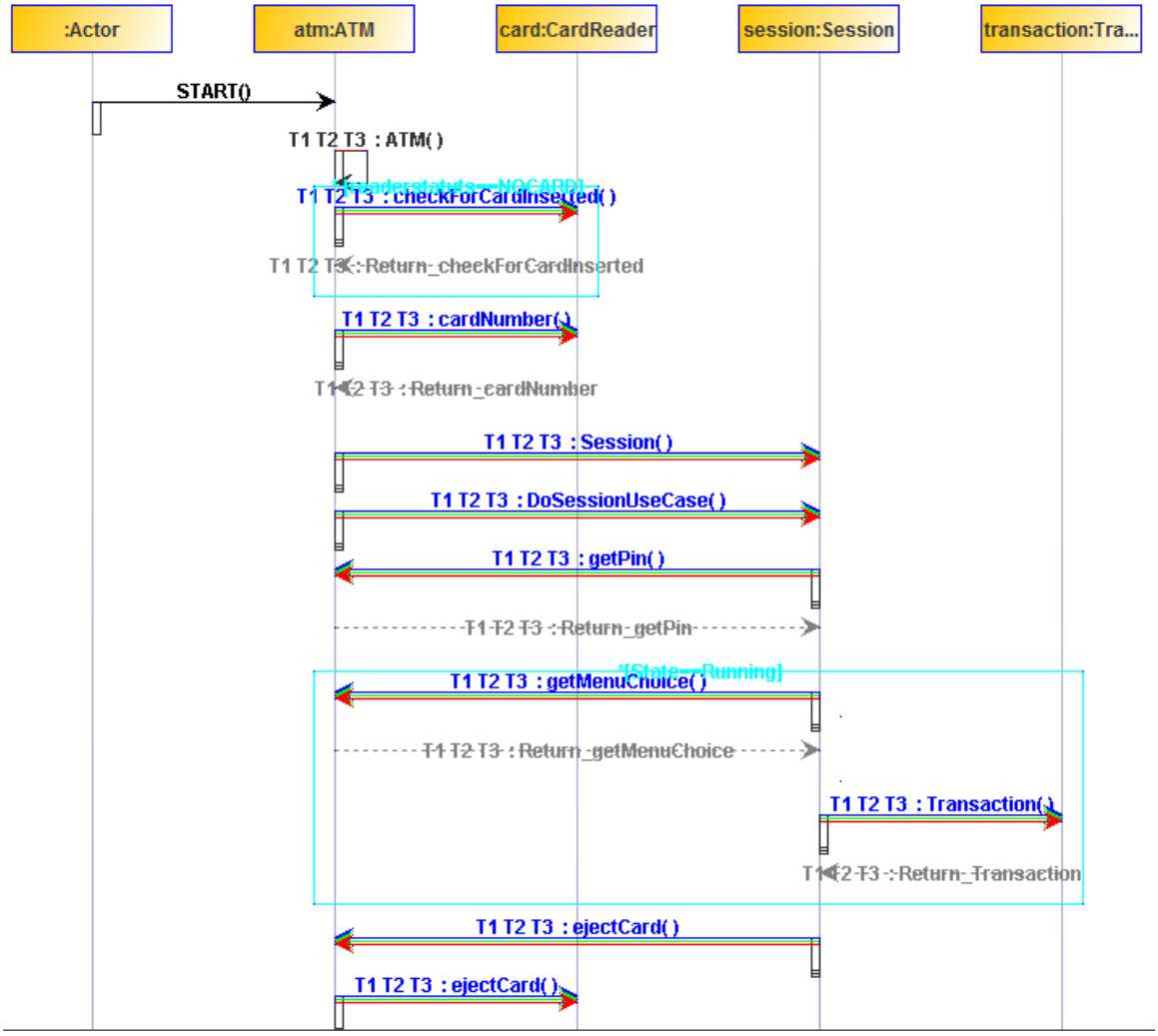


Figure A.1 Diagramme de séquence pour le scénario « *Session* » généré par le deuxième sujet.

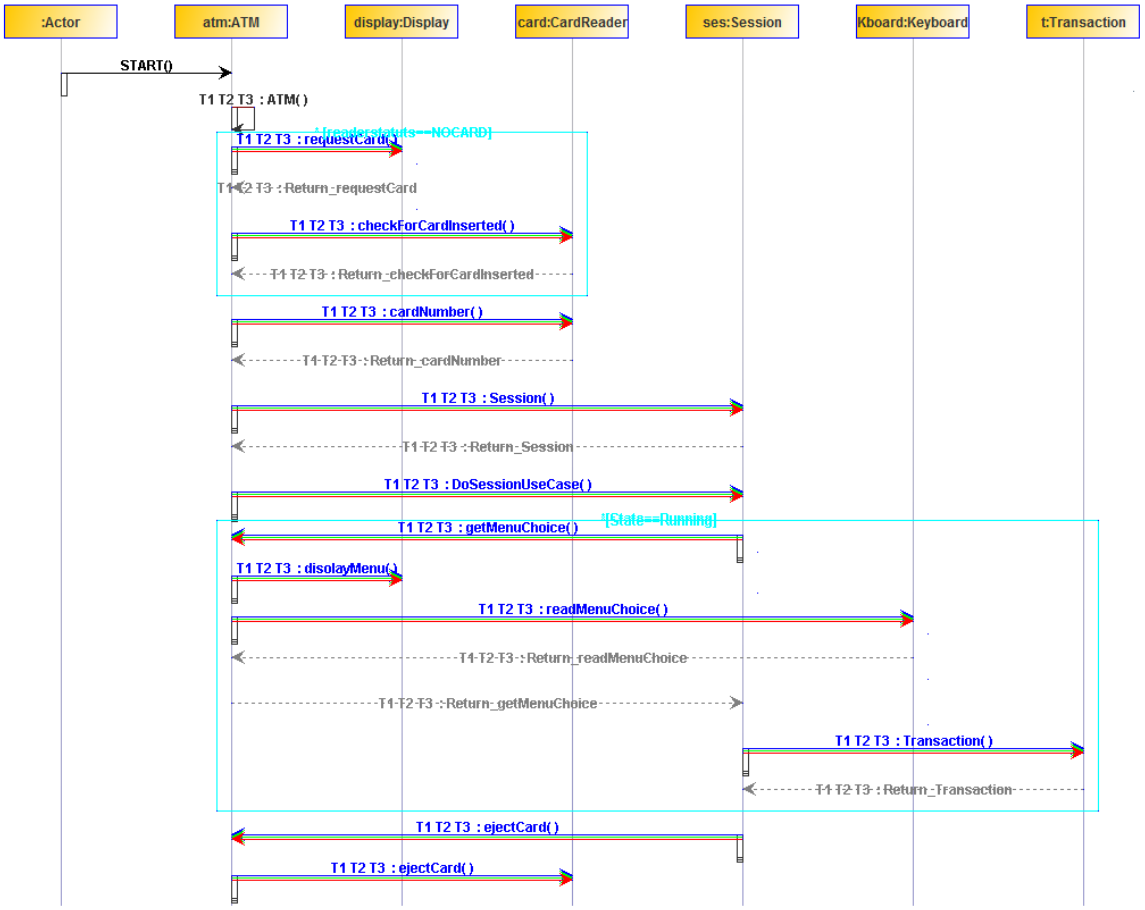


Figure A.2 Diagramme de séquence pour le scénario « *Session* » généré par le troisième sujet.

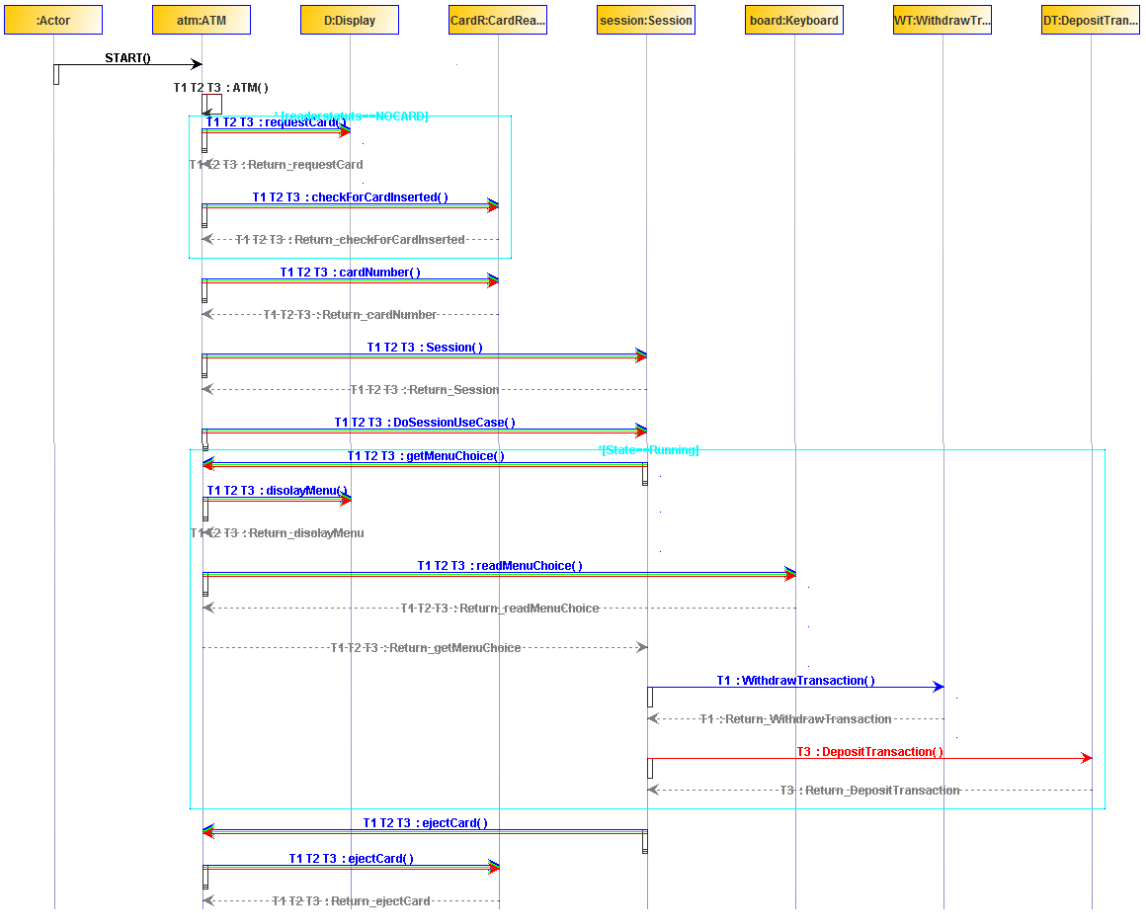


Figure A.3 Diagramme de séquence pour le scénario « *Session* » généré par le quatrième sujet.

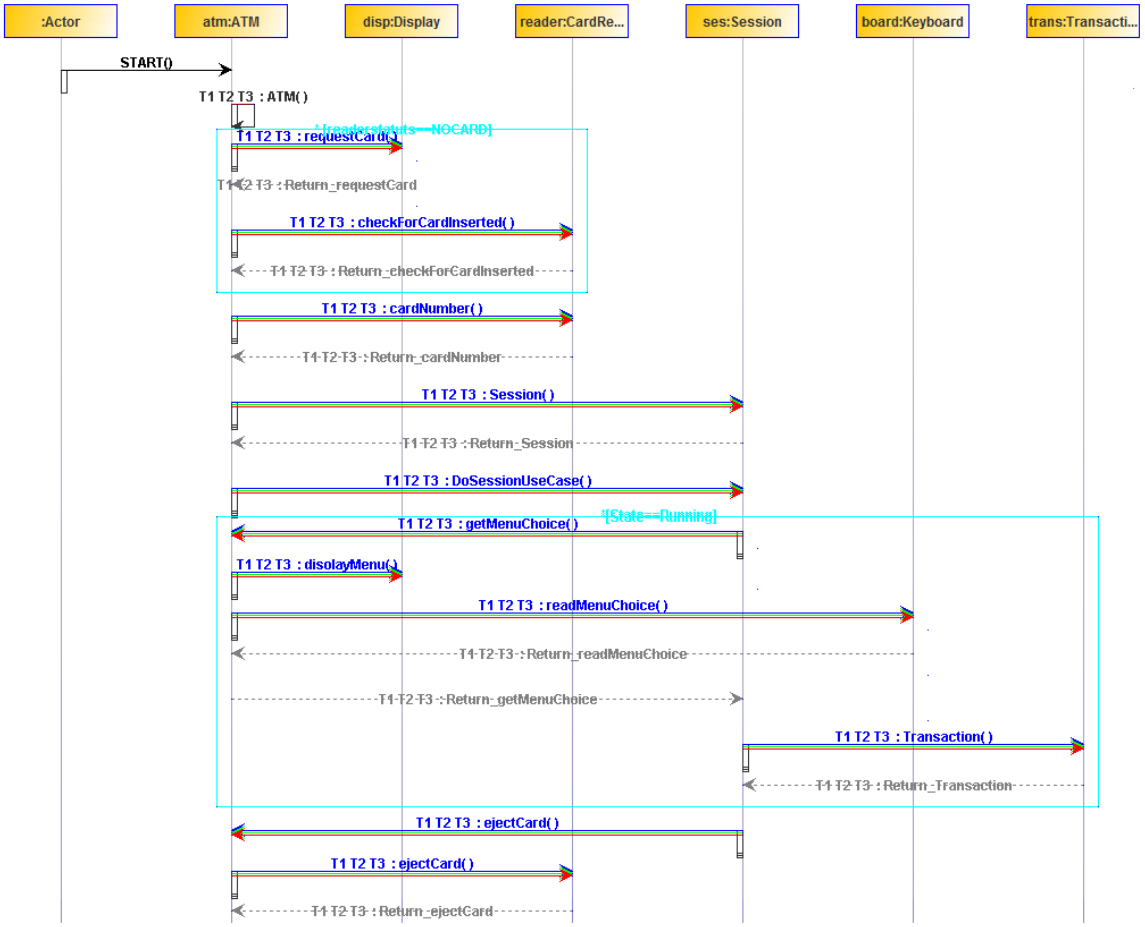


Figure A.4 Diagramme de séquence pour le scénario « *Session* » généré par le cinquième sujet.