Projet de recherche

Antoine Forgerou Jérémy Bardon Nicolas Bourdin

Sommaire

1 Introduction

Ce présent document fait office de rapport final pour le module d'initiation à la recherche effectué au sein du Master 1 ALMA. Ce module permet d'initier les étudiants au travail de recherche en informatique. Le sujet qui nous à été affectué est le suivant :

Modélisation et analyse des systèmes logiciels hétérogènes

Il s'agit la de modéliser des systèmes hétérogènes à l'aide d'automates, dans le but de les faire communiquer ensembles.

Au cours de ce rapport, différents points seront abordés. Tout d'abord une présentation du laboratoire LINA ainsi que l'équipe AeLoS que nous avons intégré le temps du module de recherche. De ce fait, nous en viendrons à expliquer la problématique du projet avec les différentes solutions adaptées à la résolution de ce dernier. Nous expliquerons donc la solution que nous avons choisi de mettre en place. Ensuite nous rentrerons au coeur du projet avec la présentation des différents outils utilisés pour mener à bien ce projet.

2 Présentation du laboratoire

Acteur central du développement de l'informatique dans la région des Pays de La Loire, le LINA (Laboratoire d'Informatique de Nantes Atlantique) est un laboratoire de recherche en sciences et technologies du logiciel qui est dirigé par Pierre Cointe. Avec ses 180 membres, ce laboratoire est actuellement situé sur deux sites Nantais : la Lombarderie (Faculté des Sciences et Techniques) et la Chantrerie (Ecole des Mines et Polytech' Nantes).

2.1 Les différentes équipes

Le LINA est composé des équipes suivantes :

— AeLoS : **A**rchitecture et **Lo**giciel **S**ûrs

— ASCOLA : **AS**pect and **CO**mposition **LA**nguages

— AtlanMod : **Atlan**tic **Mod**eling

— COD : **CO**nnaissances et **D**écision

— ComBi : **Com**binatoire et **Bi**oinformatique

— DUKe : **D**ata **U**ser **K**nowledg**e**

— GDD : **G**estion de **D**onnées **D**istribuées

— GRIM : **G**estion, **R**ésumé, **I**nterrogation, et apprentissage sur les **M**asses de données

— OPTI : Optimisation globale, optimisation multi-objectifs

— TALN : **T**raitement **A**utomatique du **L**angage **N**aturel

— TASC : Programmation par contraintes

3 L'équipe AeLoS

L'équipe AeLoS, dirigé par Christian ATTIOGBE, est une équipe de recherche intégrée au LINA résultant de la fusion de deux anciennes équipes COLOSS et MODAL.

Le projet central de cette équipe s'appuie sur les trois thématiques suivantes :

- Architecture :
- Composants logiciels corrects :
- Multiformalisme et analyse multifacette :

3.1 Les membres

Une récente liste des membres est disponible sur le site du LINA, à l'adresse suivante :
`http://www.lina.univ-nantes.fr/spip.php?page=membres&id_equipe=15&lang=fr`

4 Présentation du sujet

4.1 Problématique

Contexte : Approches formelles des systèmes embarqués communiquants.

La construction rigoureuse d'un logiciel se fait à partir de son modèle. Pour des logiciels complexes (par exemple ceux qui ont de nombreux composants différents – hétérogènes– et qui communiquent), on dispose de plusieurs modèles (hétérogènes aussi) qui doivent intérargir de façon cohérente, pour garantir l'interopérabilité sémantique entre les modèles puis les composants logiciels. Le domaine des systèmes embarqués (et aussi des objets connectés) regorge d'exemples.

Description du travail : Etudier l'interopérabilité entre des modèles de composants (logiciels ou non).
En nous appuyant dans un premier temps sur des modèles à base d'automates à états, il s'agit dans le cadre du stage d'initiation à la recherche, de contribuer à la conception et au développement d'outils passerelle entre modèles.

Nous travaillons sur plusieurs aspects :

— Lorsque cela est possible, en fonction de la sémantique des modèles, un modèle donné est encodé par un autre modèle choisi mais qui est sémantiquement équivalent.
— Lorsque cela est possible, en fonction de la sémantique des modèles, un modèle peut interagir avec un autre avec des contrats (de type Assume/Guarantee)
— Plug'N'Check (analyse systématique de modèles/composants)
— etc

4.2 Solutions proposées par l'équipe AeLoS

Pour répondre à la problématique, l'équipe AeLoS nous a proposé 3 façons de faire tout en nous laissant la liberté d'en proposer d'autres.

Dans le but de simplifier les exemples, nous avons fait le choix de représenter les modèles comme des formes géométriques que l'on essayerais d'emboiter.

La première solution consiste à adapter un des modèles au deuxième quand cela est possible. On peut prendre l'exemple du théorème de Kleene qui assure qu'un automate à états finis peut être écrit sous la forme d'une expression rationnelle et vice et versa.

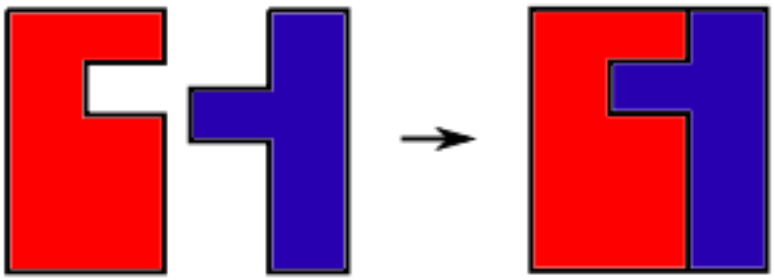


FIGURE 4.1 – Solution 1 - Adaptation d'un modèle

Dans cet exemple c'est le composant de gauche qui s'est adapté pour pouvoir intéragir avec la partie droite.

L'approche qui parait la plus évidente mais qui peut se révéler complexe consiste à intégrer un troisième modèle dans le système. Ce dernier va alors jouer le rôle de passerelle entre les deux modèles que l'on veut faire communiquer.

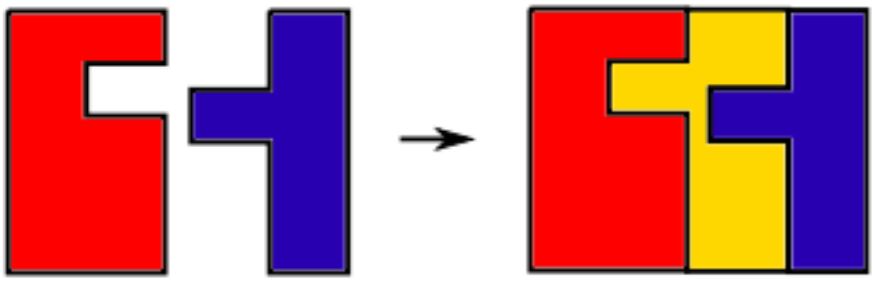


FIGURE 4.2 – Solution 2 - Interface entre les modèles

Plug'N'Check est le nom de la troisième solution que l'équipe nous a soumise. Le principe est de faire interagir les modèles ensemble puis d'effectuer des vérifications pour déterminer si les modèles peuvent communiquer à 100%, en partie ou pas du tout.

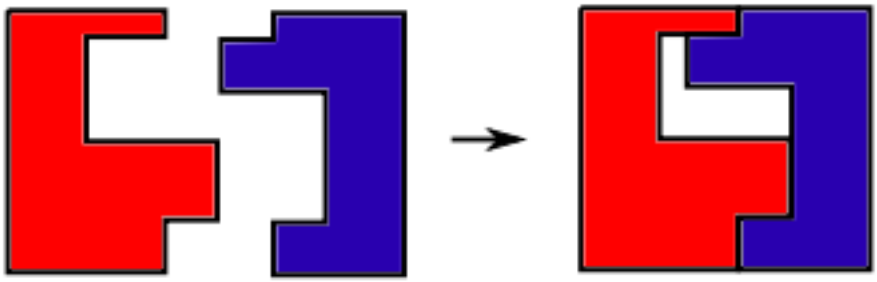


FIGURE 4.3 – Solution 3 - Plug'N'Check

L'exemple montre que certaines parties sont satisfaites mais d'autres parties ne peuvent pas communiquer.

Dans le cadre de notre stage d'initiation à la recherche, nous nous sommes orientés vers la première solution. Nous avons donc entrepris de transformer n'importe quel modèle dans un language commun afin de permettre de les interfacer.

Il est admis que les automates sont des une spécialisation de graphe, ce qui signifie que n'importe quel automate peut-être repésenté sous forme de graphe. C'est la raison qui nous a poussé à utiliser le language *DOT*

4.3 UPPAAL : un outil de modelisation des automates

UPPAAL est un environnement intégré d'outils pour la modélisation , la validation et la vérification des systèmes temps réel modélisés comme des réseaux d' automates.

Uppaal se compose de trois parties principales :

- un langage de description
- un simulateur
- un vérificateur de modèle

Le langage de description est un langage de commande non-déterministe avec des types de données (par exemple des entiers , des tableaux, etc.) . Il se sert de la modélisation ou du langage de conception pour décrire le comportement du système grâce à des réseaux d'automates étendus avec des variables d' horloge et des données .

Le simulateur est un outil de validation qui permet l'examen de possibles exécutions d'un système au début de la conception(ou la modélisation) et fournit un moyen peu coûteux de détection de défaut avant la vérification par le vérificateur de modèle qui couvre l'ensemble des comportement dynamique du système.

Le modèle orthographique peut vérifier les propriétés invariantes et l'accessibilité en explorant l'espace et l'état d'un système, c'est à dire l'analyse de l'accessibilité en termes d'états symboliques représentés par des contraintes .

4.4 Le format DOT

Le langage DOT est un langage de description de graphe dans un format texte. C'est une manière simple de décrire des graphiques que les humains et les programmes informatiques peuvent utiliser. Les graphes DOT sont généralement des fichiers avec pour extension un .gv (ou .dot).

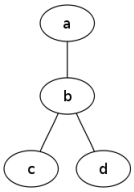
La syntaxe DOT peut aussi bien décrire des graphes non-orientés que des graphes orientés, comme des automates finis. Il est possible de placer des labels sur les transistions pour par exemple leurs donner un nom ou une explication. Le language possède aussi des attributs permettant une description graphique

plus poussée, par exemple choisir la couleur d'un noeud ou de dessiner une transition en pointillé.

4.4.1 Graphe non-orienté

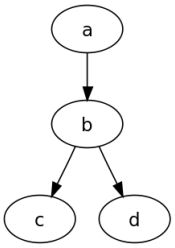
```
graph monGraphe {
    a -- b -- c;
    b -- d;
}
```

Resultat :



4.4.2 Graphe orienté

```
graph monGraphe {
    a -> b -> c;
    b -> d;
}
```



5 Hetersys [1] : notre solution

5.1 Introduction

Notre but est de pouvoir composer n'importe quel type d'automate à l'aide du logiciel UPPAAL. Celui-ci propose de créer des automates et de les rassembler sous forme de projets mais il faut obligatoirement dessiner ces automates avec les outils mis à disposition.

Etant donné que les automates sont un sous-ensemble des graphes, ce logiciel propose d'importer directement un automate – décrit sous forme de graphe – dans un projet UPPAAL pré-existant.

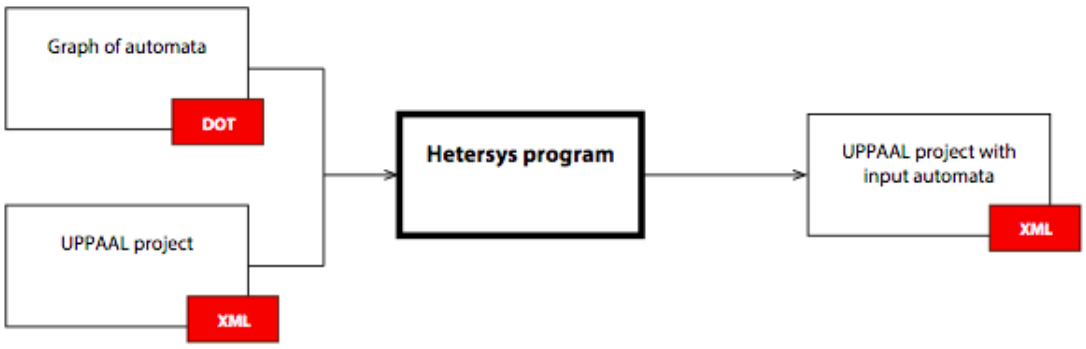


FIGURE 5.1 – Entrées et sorties du programme

Pour la description du graphe, nous avons choisi de nous appuyer sur le format DOT bien connu qui est selon nous un langage très simple et minimaliste qui permet de construire des graphes très rapidement.

1. Heterogeneous systems : Tiré du nom du sujet de recherche "Modélisation et analyse des systèmes logiciels hétérogènes"

5.2 Architecture logicielle

L'architecture interne du programme a été pensée pour être la plus modulaire possible ce qui signifie que l'on peut facilement importer et/ou exporter dans un autre format sans changer ce qui existe déjà.

En effet, le programme possède une représentation interne de l'automate sous forme de graphe ce qui rend ce genres d'ajustements plus simples. Les parties *importer* et *exporter* constituent donc les points d'extension de notre programme.

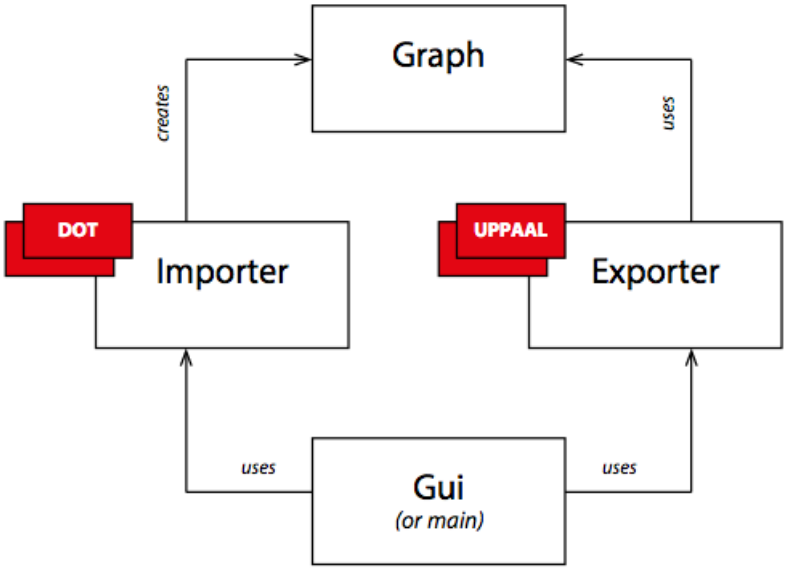


FIGURE 5.2 – Composants du logiciel

Nous avons mis à disposition une interface graphique mais nous l'avons conçue dans l'optique de la découplée le plus possible du moteur de l'application. Il serait alors aisé de la supprimer ou de la remplacer puisqu'elle suit le patron de conception MVC.

Par conséquent, la routine principale du programme ce trouve au niveau du *modèle* qui se charge de dérouler le programme tout en interagissant avec l'interface.

5.3 Fonctionnement interne

La partie principale du programme est celle qui connaît et utilise l'importeur et l'exporteur. Le fonctionnement ce résume donc à utiliser ces deux composants afin de charger l'automate et de l'ajouter dans le projet UPPAAL tout en procédant à certaines vérifications au préalable.

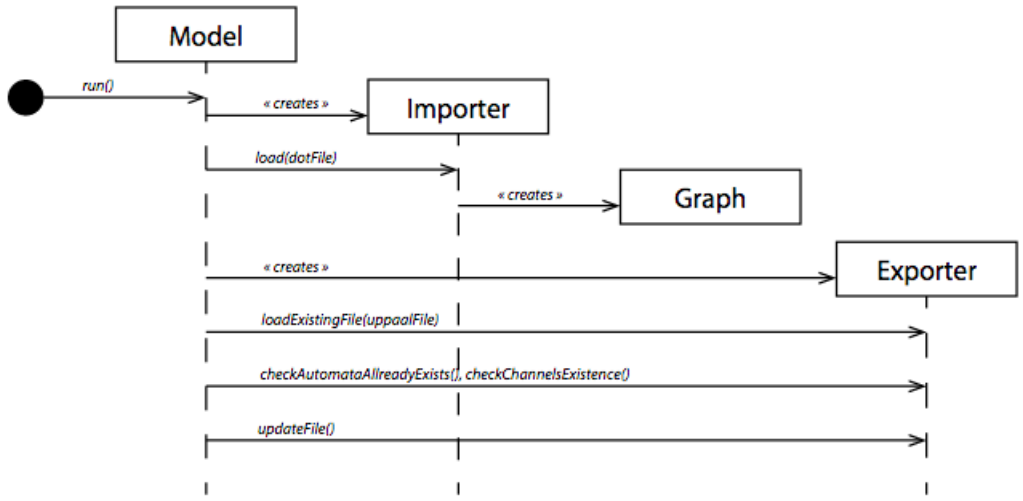


FIGURE 5.3 – Schéma du déroulement du programme (non formalisé)

La ressource la plus importante est le graphe car il est généré par l'importeur puis utilisé par l'exporteur – non représenté mais passé à la création – pour vérifier la compatibilité avec le projet UPPAAL et la conversion dans le bon format.

Par ailleurs, au niveau de l'implémentation – avec l'interface graphique – nous avons découpé cette routine en étapes. Ceci nous a permis de pouvoir reprendre à n'importe quel moment si il y a eu un souci et que l'utilisateur est en moyen de le résoudre via l'interface.

5.4 Choix techniques

Technologie

Nous avons décidé d'utiliser la technologie JAVA pour développer notre solution car il s'agit celle qui a été utilisé par les développeurs du projet UPPAAL.

Celui-ci n'est pas open source mais s'il le devient ce sera alors plus facile d'intégrer notre solution d'import directement en interne.

Aspect mis de coté

Afin de nous concentrer sur la conversion du graphe et l'importation dans le projet UPPAAL, le logiciel ne prend pas en compte la disposition graphique des éléments de l'automate.

Il se contente alors de les afficher en ligne afin de pouvoir les identifier facilement mais l'utilisateur devra les réorganiser lui-même sachant que UPPAAL n'intègre pas de fonctions de réorganisation automatique

5.5 Documentation du logiciel

5.5.1 Fenêtre principale

Il s'agit du premier élément auquel l'utilisateur est confronté mais l'interface a été pensée pour faciliter l'apprentissage.

En effet elle est décomposée en 2 parties : *Resources* et *Automata*.

La première partie regroupe les fichiers dont le programme à besoin en entrée à savoir un automate – décrit sous forme de graphe en DOT – et un projet UPPAAL pré-existant.

La seconde section s'intéresse à l'automate qui sera ajouté dans le projet. Dans un projet UPPAAL, chaque automate a un nom pour pouvoir l'identifier de manière unique ce qui veux dire que deux automates ne peuvent pas avoir le même nom. C'est pourquoi le bouton « Show list » permet d'afficher le nom des automates déjà présents dans le projet afin de ne pas les réutiliser.

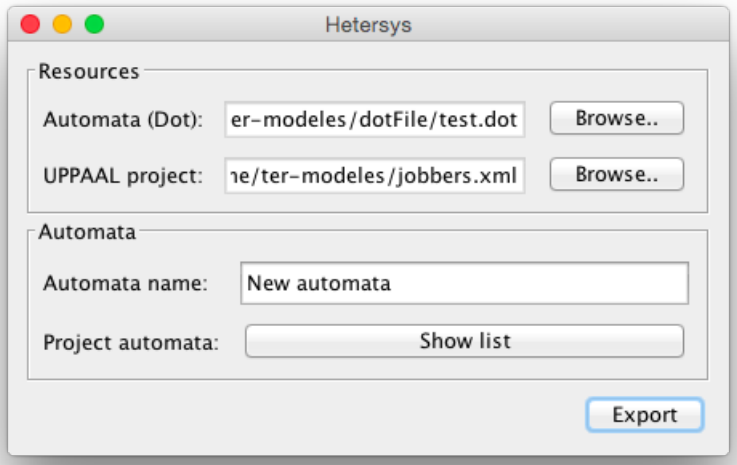


FIGURE 5.4 – Fenêtre principale

Un appui sur le bouton export permet simplement d'exporter l'automate – au format DOT – dans le projet UPPAAL donné.

5.5.2 Gestion des canaux

L'automate donné en entrée n'ayant pas de contraintes particulières – si ce n'est respecter le format DOT – il n'est pas obligatoire que celui-ci ai des canaux en commun avec les autres automates du projet.

C'est pourquoi, le logiciel va se charger de scanner les canaux utilisés dans l'automate et dans le projet pour notifier l'utilisateur dans 2 cas particuliers.

Canaux non déclarés

Dans le cas où l'automate en entrée utiliserait des canaux qui ne sont pas déclarés dans le projet UPPAAL, le logiciel demandera à l'utilisateur s'il veux les importer.

La liste des canaux permet de décider au cas par cas quel canaux il faut importer ou pas.

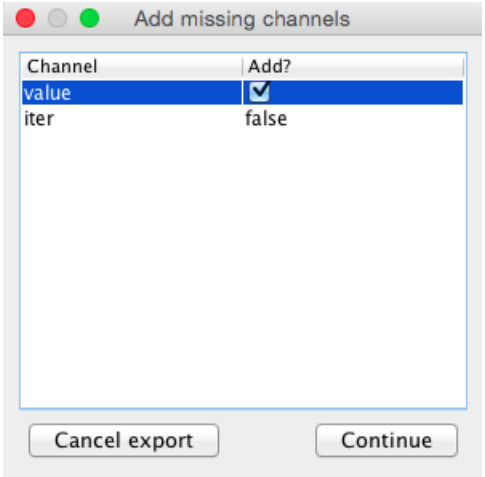


FIGURE 5.5 – Ajout de canaux

Aucun canaux en commun

Le but de ce logiciel étant de composer les automates entre eux, lorsque l'automate en entrée n'a aucun canal en commun avec ceux dans le projet un message d'alerte apparaît.

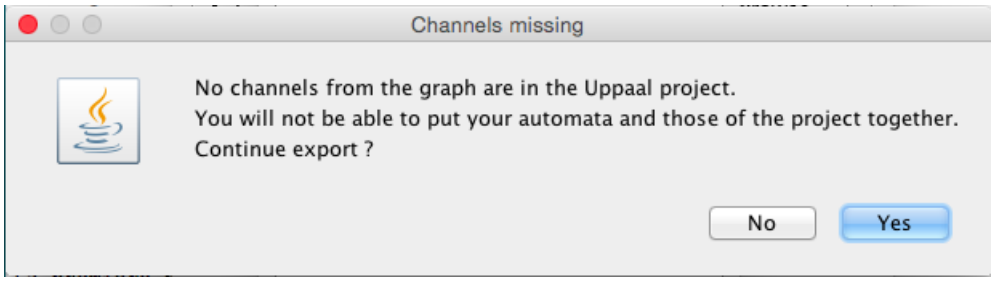


FIGURE 5.6 – Aucun lien entre les automates

Ceci permet à l'utilisateur d'annuler l'export pour changer son automate et utiliser des canaux déjà existants avant de le ré-exporter.

5.5.3 Messages d'information

Automate en double

L'automate à insérer ne doit pas porter le même nom que ceux déjà dans le projet donc une fenêtre notifie l'utilisateur si c'est le cas.

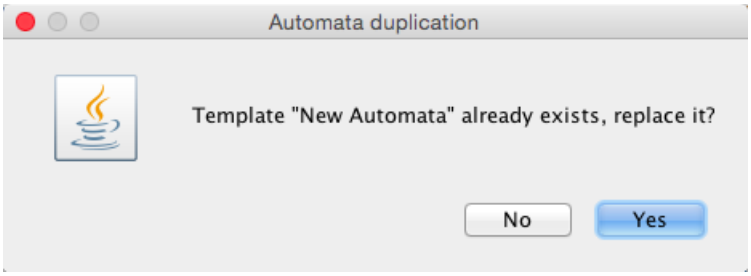


FIGURE 5.7 – Automate en double

Fin de l'export

L'export d'un automate est très rapide mais le logiciel notifie quand même l'utilisateur quand celui-ci est terminé.

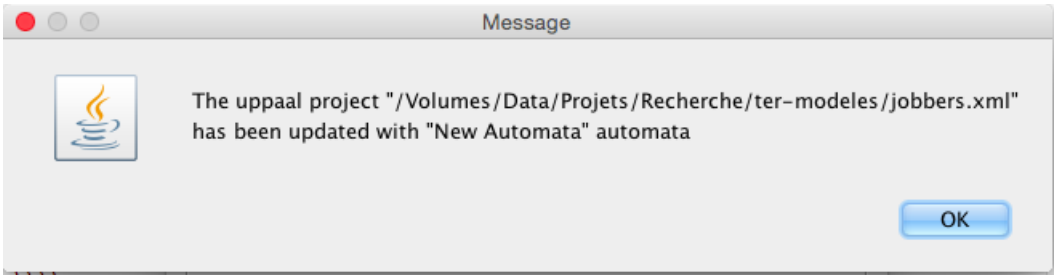


FIGURE 5.8 – Export réussi

Le message rappelle quel projet UPPAAL a été modifié pendant l'opération mais aussi le nom de l'automate qui a été ajouté.