Rapport de Stage - SimGrid Eclipse Plug-in

MERCIER Michael

RICM 4 - 2012

POLYTECH GRENOBLE

Table des matières

1 Contexte du stage

Ce stage stage de fin de 4ème année d'école d'ingénieur à Polytech Grenoble s'est effectuer dans le Laboratoire d'Informatique de Grenoble (LIG) au sein de l'équipe MESCAL (Middleware Efficiently SCALable). Cette équipe est composée de membres provenant de laboratoires, d'écoles supérieur et d'Universités (CNRS, INPG, INRIA et UJF). Le laboratoire du LIG est situé à Monbonnot-Saint-Martin près de Grenoble dans le pôle d'activité Inovallée. Mon maître de stage, Laurent Bobelin est membre de l'équipe MESCAL et travail essentiellement sur un le logiciel de simulation développé par cette équipe : SimGrid.

1.1 SimGrid

SimGrid est une boîte à outils fournissant un noyau de simulation pour les systèmes distribués dans un environnement distribué hétérogène. Le but de ce projet est de facilité la recherche dans le domaine du parallélisme et des systèmes distribués à grande échelle. Il est à la fois précis dans ses résultats et performant car il permet de simuler jusqu'à 2 millions de machines sur un seul ordinateur. Les principale force de ce projet sont :

Le passage à l'échelle Comme expliqué plus haut il peut simulé de très large système mais fonctionnent aussi très bien sur de tout petits.

Un modèle validé Dans la simulation la cohérence des résultats dépend entièrement du modèle utilisé. Celui de SimGrid a été validé théoriquement et expérimentalement.

La portablilité Utilisable sur Linux, MacOS et Windows, SimGrid permet aussi aux utilisateurs d'utiliser plusieurs langages : C et Java.

Du code Open source SimGrid est distribué sous la licence LGPL. Il est donc librement utilisable et modifiable.

Le projet SimGrid à démarré en 2010 et est toujours très actif. Depuis sa création plus de 100 publication scientifique sont basées sur SimGrid. Enfin, plusieurs outils venant de différent contributeurs permettent d'augmenter les fonctionnalités de SimGrid. Cependant, l'utilisation de SimGrid passe par l'édition de fichiers textes décrivant les entrées du simulateur. Cette édition pouvant être laborieuse et peu intuitive, il y avait donc un besoin pour un créateur de configuration et un éditeur simplifiant la création de ces fichiers. Le sujet du stage est née de ce besoin.

1.2 Sujet du stage

Le sujet de mon stage est de créer, sur la base d'un plug-in Eclipse (voir ci-dessous), une application comprenant les éléments suivants :

- un éditeur de graphique pour les fichiers XML (eXtensible Markup Language) décrivant la plate-forme du réseau utilisée par SimGrid pour la simulation. Ce fichier décrit topologie ainsi que le routage du réseau à l'aide de balise décrite dans un fichier de grammaire de type DTD (Document Type Definition).
- un assistant de création de projet afin de générer tous les fichiers ainsi que la configuration nécessaire à l'utilisation de SimGrid.

Cet outils est destiné à visualiser et éditer des plate-formes existantes, à permettre une prise en main rapide de SimGrid et à faciliter son utilisation pour tout les d'utilisateurs.

1.2.1 Eclipse

FIGURE 1 – logo Eclipse

Eclipse (Figure 1)est un Environnement de Développement Intégré (EDI). C'est un outil permettant l'édition la compilation et le lancement de code source. Il à la particularité de permettre son extension par un système de plug-in inter-dépendants. Il est donc possible d'étendre les fonctionnalités de plug-in existants et d'offrir des extensions pour les autres plug-in. C'est ce mécanisme qui est utilisé par l'application développer lors de ce stage.

1.2.2 EMF/GEF ?

Le sujet du stage ne défini pas avec quelle outils le plug-in doit être réalisé à l'intérieur d'Eclipse. Cependant il suggère l'utilisation de la combinaison de deux framework Eclipse fortement combiné ensemble : Eclipse Modeling Framework (EMF) et Graphical Editing Framework (GEF). EMF est utiliser pour générer un modèle de donnée à partir des donnée passer en paramètre (un fichier de Grammaire XML Schema par exemple). Ce modèle est ensuite utilisé par GEF pour la création d'un éditeur graphique. Bien que ces deux framework soit souvent utilisé ensemble, GEF accepte tous type de modèle. L'utilisation de ce couple d'outils doit donc être déterminé lors du stage.

2 Travail réalisé

2.1 Découverte des différents frameworks

Le choix des outils n'étant pas prédéterminé, il a fallut passé par une phase de découverte des différents frameworks que propose Eclipse pour développer des plug-in. L'utilisation de GEF semble être indispensable mais, comme expliqué plus haut, le choix du modèle reste à déterminer. De plus il existe d'autre outils plus haut niveau, comme Graphiti basé sur EMF/GEF qui génère très rapidement un éditeur graphique utilisable mais peu configurable. Il est apparu que cet outils était en phase d'incubation, donc peu fiable, et qu'il ne permettait pas un configuration suffisante pour implémenter toute les fonctionnalités nécessaires.

2.1.1 Choix du modèle

Le problème du choix du modèle utilisé est complexe. Le framework EMF permet de générer un modèle à partir d'un fichier de grammaire XML schema or SimGrid utilise le format DTD. Une conversion est malheureusement impossible car le format XML Schema est moins permissif et l'accepte pas ce qui est autorisé dans la DTD de SimGrid. Nous avons ensuite penser à l'utilisation de JAXB qui un outils permettant la génération et la liaison d'un modèle java avec un fichier XML. Mais notre éditeur de plate-forme doit aussi comporter un éditeur texte XML, qui maintien lui aussi une structure de données lié au fichier. L'accès à ce modèle étant possible, son utilisation pour l'éditeur graphique permet de ne maintenir qu'un seul modèle qui est de plus déjà intégrer dans l'environnement d'Eclipse et maintenu par le l'éditeur texte. Le choix s'est finalement porté sur une implémentation direct avec GEF en utilisant le modèle de l'éditeur de texte nommé DOMModel.

2.1.2 Utilisation modèle MVC de GEF

Le framework GEF est basé, comme beaucoup d'outils permettant de faire des interfaces graphiques, sur le concept de Modèle Vue Contrôleur (MVC). Cela permet la séparation de la couche Modèle qui contiens les données, les Vues qui représente l'affichage, et les Contrôleurs qui gèrent les actions de l'utilisateur et du programme. Le modèle à été décrit au-dessus, et dans GEF les contrôleurs sont des "EditParts" organiser en hiérarchie dans le "GraphicalViewer" lui même contenue dans l'éditeur graphique.

2.2 Création du noyau fonctionnel

Le noyau fonctionnel, c'est à dire les fonctions de bases permettant toute la gestion du modèle ainsi que des règles métiers internes de l'application. Le modèle n'étant pas directement détenu par notre plug-in il n'a pas été possible d'étendre les classes du modèle afin d'ajouter les fonctionnalités qui nous sont nécessaires.

2.2.1 ModelHelper

Une bibliothèque de méthodes statiques permettant l'accès, l'ajout, la suppression et l'édition d'éléments nommé "ModelHelper" à donc été créé. Elle est utilisée comme utilitaire par les contrôleurs pour centraliser les accès au modèle : cela limite les dépendances au modèle modèle extérieur et seul cette classe devra être modifiée en cas de changement de celui-ci.

2.2.2 SimgridRules

Cette classe contient toute les règles métiers de SimGrid. L'accès à ces règles se fait au travers de méthodes statique revoyant vrai ou faux.

2.2.3 ElementList

Cette classe est un singleton, c'est à dire qu'elle est instanciée une seul fois. Elle contient toutes les constantes correspondant au noms des différents éléments ainsi que des méthodes d'accès aux attributs de chacun de ces éléments à l'aide d'un parseur de DTD.

2.3 Gestion de la création et édition d'éléments graphiques

2.3.1 Création générique des éléments graphiques

La création des élément graphique se fait selon les principe de GEF à travers une classe usine qui génère les élément en fonction du modèle et de contexte. J'ai donc crée une classe usine qui récupère l'étiquette de la balise XML afin d'inférer la classe de l'"EditPart" associé. Si la classe est trouvé l'élément est crée, sinon il est ignoré. Cela permet de rajouter simplement des élément en ajoutant simplement une classe ayant le bon nom dans le bon package, sans avoir à toucher à l'usine.

2.3.2 Création des liens

2.3.3 Gestion de la position

GEF utilise le modèle pour se mettre à jour. Or la position des éléments graphiques n'étant pas contenue dans ce modèle il a fallut implémenter un classe reliant chaque élément du modèle à sa position. C'est un singleton qui conserve les positions tout au long de la session. Ces position ne sont pourtant pas persistante d'une session à l'autre pour évité l'incohérence si le fichier XML est éditer en dehors du plug-in. Une fois la persistance de la position des éléments, il faut les mettre en page correctement. (**fiche suivie 1**)Ajout de l'action auto-layout avec graphstream

2.3.4 Création d'assistant de création/édition d'élément

Pour les éléments complexes, nécessitant plus qu'un simple glisser-déposer pour leurs créations, j'ai développé un ensemble de fenêtres assistants l'utilisateur dans cette tâche. L'assistant est générique et ajoute les pages nécessaires en fonction du type d'élément. La création des routes et des clusters en particulier a nécessité l'implémentation d'assistant de plusieurs pages. Vous pouvez voir ces pages en annexe.

2.4 Implémentation des politiques de gestion des commandes

Dans le framework GEF chaque contrôleur implémente une politique de gestion des commandes. Cette politique définie de quelle manière l'élément associer à ce contrôleur doit réagir au demande envoyer par l'utilisateur ou par le programme lui-même, lors d'une mise à jour par exemple. Elle interdit ou autorise la création de commandes. Ces commandes envoyées dans la pile de commande de l'éditeur puis exécutées dans l'ordre de leurs arrivées. Chaque commande contient une méthode "execute()" qui effectue des modifications sur le modèle qui sont ensuite répercutées sur la vue. Ces commande contiennent aussi les méthodes "undo()" et "redo()" qui permet à l'utilisateur naviguer dans la pile de commande.

2.5 intégration dans Eclipse

2.5.1 OSGi

Eclipse à plusieurs mécanismes internes qui permettent d'étendre ces fonctionnalités. La base du mécanisme de fonctionnement des plug-in est le framework OSGi (Open Services Gateway initiative). Il gère le cycle de vie et les dépendances des plug-in à l'intérieur d'Eclipse. Chaque plug-in est encapsulé dans un composant nommé "bundle". Un "bundle", requière et expose des dépendances avec d'autres "bundles" qu'il à fallut gérer correctement pour qu'il fonctionne lors du déploiement. Les bibliothèques qui n'était pas des "bundles" qui ont été intégrer au plug-in, ont elles aussi nécessité une configuration particulière.

2.5.2 Création d'un aperçu

Eclipse contient par default plusieurs vue destiné à aider le navigateur dans son utilisation des éditeurs. Pour une bonne intégration dans Eclipse, il faut implémenter certaines de ces vues. C'est pourquoi la vue "aperçu" contenant une arborescence de la plate-forme réseau à été ajoutée. Elle permet à la fois d'apercevoir la structure du réseau et de facilité la navigation en permettant la sélection des éléments.

2.5.3 Synchronisation de la sélection

La synchronisation de la sélection en fonction des différentes vue est une partie très délicate. Elle est traité plus en détails dans dans la partie 6.

2.6 Création de projet assisté

Afin que le plug-in soit complet il se devait de gérer des projet dédiés à SimGrid contenant tous les fichiers nécessaires, et étant configuré pour une utilisation simple.

2.6.1 Projets Java et C

Deux type de projet SimGrid ont été implémentés. Ils comprennent le fichier de plate-forme relié à l'éditeur ainsi que des fichiers qui sont générés en fonction de l'environnement de programmation et du langage choisi par l'utilisateur pour coder sa propre simulation au dessus de SimGrid. Deux langages sont disponibles : C et Java. La génération de ces fichiers est relativement simple, et la complexité de cette tache s'est retrouvée dans la création et la configuration du projet lui-même : Comprendre les structures internes qui représentent chacun des types de projets (C et Java) pour pouvoir les configurer correctement pour l'utilisation de SimGrid quelque soit le système d'exploitation, la version d'Eclipse et la version du simulateur lui-même.

2.7 Documentation

Tout projet se voulant utilisable et maintenable contient une documentation utilisateur et développeur.

2.7.1 Site utilisateur

Le choix à été pris de faire un site permettant à l'utilisateur de trouver l'outil et la documentation pour apprendre à l'installer et à l'utiliser

2.7.2 Wiki développeur

La documentation à pris la forme d'un wiki. C'est la forme la plus adapter pour une documentation destiné au développeur, car elle permet à tout les contributeurs (ce projet étant Open Source) de participer à son évolution. Il contient un lien vers la traditionnelle Javadoc qui est généré à partir des sources dont toute l'API public à été clairement commenté.

2.8 Site de dépôt

J'ai finalement mise en place un dépôt public sous forme d'un site de mise a jour Eclipse pour permette aux utilisateurs d'installer et de mettre à jour le plug-in facilement. Des tests on été effectué sur différent système et différente version d'Eclipse pour s'assurer du bon fonctionnement du Plug-in.

3 Bilan et perspectives

3.1 prise en main des APIs d'eclipse

3.1.1 OSGi

3.1.2 GEF

gestion des différence modèle/affichage

3.1.3 synchronisation de la sélection

Ce plug-in utilise comme conteneur pour les deux éditeurs (graphique et textuel) un éditeur multi-page. Une des difficultés rencontrées est synchronisation de la sélection entre les deux éditeurs au sein d'un « multi-éditeur », ainsi que la mise à jour des action disponibles en fonction de cette sélection. Ceci est du au faite que l'utilisation du « multi-éditeur » est mal documentée et reconnue pour son fonctionnement complexe et une grande difficulté d'utilisation.

3.2 SWT et les wizard

Une des difficultés rencontrées, a été la création de de fenêtres d'assistances (« Wizard ») pour la création/suppression/modification d'éléments ainsi que la création de projets SimGrid. Il m'a fallu passé par l'apprentissage de la bibliothèque graphique SWT utilisé par Eclipse qui est assez complexe, notamment au niveau des « layout » qui permettent la gestion de la mise en page.

3.3 création des projets

Le manque d'exemple et de documentation pour la partie CDT (C Developement Tools) a nécessité l'exploration approfondit du code. La configuration correct du projet C n'a pu aboutir que par tâtonnement à l'aide des outils de Debug.