Master research Internship

ENSSAT
LANNION

Bibliographic report

Vérifiaction de systèmes interactifs critiques par le raffinement et la preuve à l'aide d'Event-B

Domain: Formal Languages and Automata Theory - Distributed, Parallel, and Cluster Computing

Author:
Romain geniet

Supervisor:
Yamine Ait-ameur
ACADIE

Abstract: Ce document présente deux approches essentielles de la vérification formelle de systèmes distribués : la conception de système interactifs utilisant une approche correct-by-construction et l'utilisation de technique formelle utilisant le raffinement et la preuve. Le but est de permettre la création d'un langage de spécification modélisant la construction de système valide par l'approche correct-by-construction.

Contents

1 Introduction

Un système est un ensemble d'éléments indépendants de natures diverses qui opèrent tous dans un même cadre. La nature de ces éléments peut être électronique, mécanique, logicielle voire même humaine. L'ingénierie de systèmes a pour but de formaliser et d'appréhender la conception de systèmes complexes avec succès. L'objectif est de vérifier le bon comportement des systèmes créés afin qu'ils puissent être une réponse fiable à une demande spécifique. Cette approche est utilisé lorsque le système est trop complexe pour que les méthodes de pilotage standards soient utilisées.

Dans le cadre de la conception formelle de système, plusieurs approches permettant le développement de systèmes ont montré que les méthodes formelles sont utilisable efficacement et de manière adaptative. De plus, l'utilisation de ces approches prend en compte les cycles de vie des composants physiques des systèmes créées - dans des domaines comme l'aéronautique et aéro-spatiale ou encore le secteur médicale par exemple.

L'utilisation de l'ingénierie de systèmes requiert, en particulier, l'adaptativité des méthodes de création - par exemple la capacité pour un système situé dans une fusée spatiale de fonctioner quelque soit la pression. Pour ce faire, il est donc nécessaire de créer des systèmes robustes, adaptatifs et sûr pour répondre aux exigences du cahier des charges donné à la création de tels systèmes.

Ce document est centré sur le thème de la preuve formelle dans le domaine des systèmes distribués. En effet, le domaine étudié se compose de deux branches. La première est la conception de systèmes interactifs utilisant l'approche correct-by-construction. La seconde est l'utilisation de techniques formelles basées sur le raffinement et la preuve.

La première étape de l'activité liée à ce document est le processus de développement formel. La manière canonique de procéder est la définition d'un système interactif au travers d'échanges de messages entre les différentes composantes du système distribué que nous étudions. Ensuite, nous synthétisons les comportements des différents systèmes qui échangent entre eux. Pour cela, nous utilisons la méthode Event-B.

La seconde étape consiste à mettre en place un langage de spécification à partir des observations faites lors de la première étapes. Un tel langage permettrait de concevoir des systèmes intéractifs de manière incrémentale et correct par construction grâce à la méthode prouvée ci-avant.

Pour ce faire, nous commençons par définir les différents outils utilisés lors de la preuve formelle et la preuve par raffinement. Ensuite, nous présentons les enjeux et outils de la spécification et de l'aproche correct-by-construction.

2 Event-B et la preuve formelle

Dans cette première partie, nous présentons les outils et les enjeux de la vérification de systèmes interactifs critiques.

2.1 Preuve formelle : les enjeux [15], [16]

Commençons par définir les enjeux de la preuve formelle. Certains programmes ou systèmes informatiques sont plus critiques que d'autres. Par exemple, un pilote automatique d'avion doit obligatoirement être fiable à cause du nombre de vies en jeu lors de l'exécution alors qu'un programme donnant le taux d'humidité dans une pièce est moins critique.

Ainsi, il est donc nécessaire de prouver formellement le bon fonctionnement de certains programmes. Pour cela, il existe plusieurs approches possibles.

Ces approches, ou méthodes formelles, sont basées sur la logique mathématique et la théorie de la calculabilité. Elles ont pour objectif, dans le contexte informatique, de vérifier la validité d'un programme par rapport aux exigences du cahier des charges. Ainsi, les différentes approches dans le domaine de la vérification et la preuve formelle consiste à construire un système sûr et fiable. En effet, la preuve d'un programme par une méthode formelle garantie - avec une probabilité d'erreur très faible - l'absence d'erreur à l'execution. La preuve automatique de théorèmes permet de laisser le système se construire à l'aide d'axiomes à prouver. Cependant, certains problèmes se révèlent indécidables. Une autre approche est celle du model checking. Cette méthode consiste à simuler l'évolution du système durant l'exécution du système. En fonction de l'espace occupé et de la taille du graphe d'état atteignable, cette méthode peut être extrêmement coûteuse. Pour les besoins du stage, la méthode du model checking n'est pas utilisée. Les preuves demandées sont basées sur la communications des composants de systèmes distribués. Pour répondre à la problématique de la communication de système distribués, nous utilisons la méthode Event-B. La méthode Event-B est une approche de preuve formelle basée sur les évènements.

2.2 Event-B [1],[4]

Event-B est une méthode formelle de modélisation et d'analyse au niveau système. Cette méthode fonctionne par le biais de la théorie des ensembles (en tant que théorie de modélisation), l'utilisation des raffinements afin de modéliser les différents niveaux d'abstractions du système étudié et l'utilisation de la preuve mathématiques pour prouver la consistance des différents niveaux de raffinement.

La méthode Event-B permet de modéliser de façon abstraite le comportement et les spécifications d'un logiciel dans le langage d'Event-B -langage B. Puis, suite à des raffinements successifs, on obtient un modèle concret en langage B mais traduisible dans un langage comme Ada ou C et qui peut être exécuté par un ordinateur. Lors des raffinements successifs, des détails sont ajoutés au modèle du système afin de permettre de le rendre plus concret à chaque étape de raffinement. Une activité de preuve formelle permet de vérifier la cohérence du modèle abstrait et la conformité de chaque raffinement avec le modèle supérieur (prouvant ainsi la conformité de l'ensemble des implantations concrètes avec le modèle abstrait).

Afin de mettre en place la méthode Event-B, il existe un IDE nommé Rodin - plateforme utilisée dans le cadre du stage. Cette plateforme est un IDE proche d'Éclipse (IDE mis en place par Oracle pour différents langages de programation tels que java et C/C++). Cette plateforme permet de mettre en place la preuve mathématiques et la mise en place du raffinement. Tout comme Éclipse,

cette plateforme est opensource. Cette plateforme a été créé il y a trois ans et son but est de mettre en place la totalité de l'environnement de preuve pour la méthode Event-B.

Cette méthode a été développé en 1988 par le chercheur français Jean-Raymond Abrial - qui a développé d'abord la methode B classique. Les principaux axes de la méthode Event-B sont liés à la logique mathématiques du premier ordre et à la théorie des ensembles (comme expliqué ci-dessus). Les étapes principales de cette méthode sont la spécification des objectifs par des machines abstraites. Plus la méthode est à un stade avancée, moins le niveau d'abstraction est élevé. La dernière étape de cette méthode est l'implantation du système voulu.

Cette méthode est la conséquence des travaux de Hoare sur les pré-conditions et post-conditions et de ceux de Dijkstra. Le raffinement offert par cette méthode - permettant de décomposer le système en graphe d'état-transition et de diminuer le niveau d'abstraction en ajoutant des détails à chaque étape - est lié au travaux d'Abrial et de Hallerstede.

2.3 Langage de Spécification

Nous allons nous intéresser aux langages de spécification. Pour cela, nous allons tout d'abord présenter un langage de spécification particulier avant de présenter les caractéristiques d'un tel langage.

2.3.1 Un exemple de langage de spécification : le CSP [2]

Le langage CSP (Communicating Sequential Process) est un langage formel permettant de modéliser des interactions d'un système distribué ainsi que la description et l'implantation des paradigmes classiques de la concurrence dans de tels systèmes.

Ce langage, bien que n'étant pas un langage de programmation comme le sont le C ou le Java, est utilisé comme un outil de spécification formelle par l'industrie et a été présenté pour la premier fois en 1978 par C.A.R Hoare et est utilisé dans les champs applicatifs d'application parallèle et dans le domaine du calcul formel.

Le langage fonctionne avec les concepts classiques de typage, de conjonction séquentielle, d'affectation des valeurs et de déclaration. Cependant, d'autres concepts sont absent de ce langage (les procedures sont inexistants et une majorité des entrées/sorties usuelles sont inutilisables).

Voici un exemple d'utilisation du langage CSP dans le cadre de l'algorithme de Naimi Trehel (code pris dans le cours d'algorithmique répartie de Gwénolé Lecorvé donné à l'ENSSAT):

P[i] ::
pere : entier;
suivant : entier := nil;
– nil = rien
demande : booleen := faux ;
jeton : booleen;
– Initialisation de pere et jeton
[i = 1 →
pere := nil;
jeton := vrai
[] i =≠ 1 →
pere := 1;
jeton := faux];

– Demande d'entree en section critique par le processus de traitement
*[¬ demande ; A[i] ? besoin_SC() →
– P[i] n'est pas la racine
[pere =≠ nil →
P[pere] ! dem_SC(i) ; pere := nil
– P[i] est la racine et on a le jeton
[] pere = nil ; jeton -¿ A[i] ! debut_SC()
– P[i] est la racine mais n'a pas le jeton
[] pere = nil ; ¬ jeton →
skip];
demande := vrai
– Fin de section critique
[] demande ;
jeton ;
A[i] ? fin_SC() →
[suivant ≠ nil →
P[suivant] ! jeton() ;
suivant := nil ;
jeton := faux
[] suivant = nil → skip
];
demande := faux
– Reception d'une demande sans etre racine
[] pere ≠ nil ; P[k] ? dem_SC(j) →
P[pere] ! dem_SC(j) ;
pere := j
[] pere = nil ; P[k] ? dem_SC(j) →
– P[i] a obligatoirement le jeton s'il n'est pas demandeur
[¬ demande → P[j] ! jeton() ;
jeton := faux [] demande →
suivant := j];
pere := j
– Reception du jeton
[] P[k] ? jeton() → jeton := vrai ;
A[i] ! debut_SC()
]

2.3.2 Formalisation des langages de spécification [14]

Maintenant que nous avons présenté un langage de spécification, intéressons nous à l'ensemble de ces langages.

Les langages de spécification sont des langages formels. Commençons par définir les langages formels.

Un langage formel est un ensemble de mot sur un alphabet tel que chaque mot de ce langage est un mot (ou formule) bien formé. Une mot ou une expression bien formée est un mot (respectivement une expression) formé à l'aide d'une combinaison de règles de la grammaire formelle générant le

langage. On peut définir un langage formel de plusieurs façon : en utilisant un automate, une grammaire formelle, des expressions rationnelles, un ensemble d'instance de réponse à un porblème de décision dont la réponse est positive.

Un langage de spécification a pour but de décrire les objectifs et non les méthodes pour atteindre ces objectifs. Ainsi, ces langages - pour la plupart - ne sont pas des langages produisant des fichiers exécutables. Ces langages utilisent la théorie des ensembles ainsi que des structures algébriques et de la théorie des modèles qui sont utilisés dans l'optique de la description des objectifs.

3 Les systèmes distribués [11], [12]

Présentons maintenant les systèmes distribués et différentes méthodes de calcul et de communication sur ces derniers.

3.1 Présentation des systèmes distribués

D'un point de vue profane, un système distribué est un système non centralisé à plusieurs éléments. Le système permet des calculs plus rapides. Le calcul sur les systèmes distribués est plus rapide et plus répandu que le calcul centralisé mais il est aussi plus mystérieux - moins proche de la méthode de calcul humaine. Les différents composants communiquent entre eux.

D'un point de vue plus formel, un système distribué est un ensemble composé d'éléments reliés par un système de communication. Chaque élément possède des fonctions de traitement (processeurs), de stockage (mémoire) et de relation avec le monde extérieur. Les éléments du système ne fonctionnent indépendamment mais contribuent à une ou plusieurs tâches. Une partie au moins de l'état global est partagé par chaque composante du système.

Les calculs sur de tels systèmes peuvent prendre plusieurs formes en fonctions de l'objectifs et de la technologie utilisée. Par exemple, si une collection de machines doit modifier une donnée unique partagée, alors, il existe plusieurs méthodes afin de mettre en place un (ou plusieurs) algorithme(s) - par exemple celui de Naimi-Trehel - qui répond au besoin tout en empéchant d'avoir plusieurs accès simultanée à la même donnée.

3.2 Systèmes distribués et mémoire partagée

Dans la partie précédente, nous avons évoqué le cas de la communication par mémoire partagée. Maintenant, présentons en détail cette technique.

3.2.1 Qu'est ce qu'une mémoire partagée ?

Lorsque plusieurs des unités d'un système distribué ont besoin d'avoir accès à une même ressource, cette dernière est dite partagée. Par exemple, si plusieurs joueurs ont accès au même information dans un jeu en ligne, une partie des données est partagée par l'ensemble des joueurs. Tous les joueurs ne peuvent pas modifier la même donnée en même temps pour des raisons de cohérence dans le jeu. Un autre exemple de mémoire partagée est une donnée commune à plusieurs clients connectés à un serveur comme la température dans une maison - dans ce cas, les clients sont des capteurs automatiques.

Dans le cas des systèmes répartie, une mémoire partagée peut être vue comme un système à m producteurs pour 1 consommateur. Pour que chacun puisse avoir accès à la ressource, il convient donc d'ordonnancer les accès à la ressource.

3.2.2 Calcul en mémoire partagée

Maintenant que nous avons défini ce qu'est une mémoire partagée, présentons les calculs dans le cas de ces mémoires pour le cas des systèmes distribués.

Pour mettre en place les calculs sur la mémoire partagée, il existe plusieurs solutions possibles. Commençons par présenter le principe de l'exclusion mutuelle dans le cas des systèmes distribués. Un seul processus peut avoir accès à la ressource partagée afin de la modifier. L'accès en lecture

peut, quant à lui, concerner plusisuers processus. En effet, la lecture ne modifie pas la ressources ce qui rend le nombre de droit d'accès moins strict. Il existe trois états possibles pour chaque processus. Le processus peut être demandeur d'accès pour la ressource partagée, ou avoir accès à la ressource partagée. Le troisième état est l'état où le processus n'est pas demandeur et n'a pas accès à la ressource. Un processus peut passer de sans interraction à demandeur ou de demandeur à accès à la ressource ou encore de demandeur d'accès à sans interaction. Le système ordonnanceur permet de gérer le passage de demandeur à a accès à la ressource. Pour cela, le système ordonanceur peut être assimilé à un contrôleur. Tout algorithme d'exclusion mutuelle possède trois propriétés : la propriété de sûreté - il y a au plus un processus dans l'état accès à la ressource - celle de vivacité garantissant l'absence totale d'interblocage - un interblocage a lieu lorsque deux processus concurents s'attendent l'un l'autre - et la proprité d'équité - tout processus demandant l'accès à la ressource peut y avoir accès en un temps fini.

Il existe plusieurs types de contrôles pour les algorithme : les contrôles par jeton, par serveur ou par permission. Chacun de ces algorithmes vérifie les propriétés données ci-avant. Dans le premier cas, le jeton permet d'avoir accès à la ressource. La circulation et la gestion du jeton sont géré par des processus qui assurent la communication entre les différents composants du système. Le second cas considère que le système comporte un serveur centralisant les autorisation d'accès à la ressource. Enfin, la dernière méthode considère que les différents composants du système se donnent mutuellement l'accès à la ressource. Dans chaque cas, il existe plusieurs algorithmes permettant de gérer les accès.

Présentons maintenant la technique de distribution par éclatement. Cette technique consiste à utiliser un tampon de n cases pour m producteurs et un consommateur. L'idée est de diminuer les contraintes. L'accès à la ressource est donnée en fonction des productions en cours et des demandes effectuées par les processus. Cette technique est bien moins efficace que l'exclusion mutuelle présenté ci-avant (cours de Gwénolé Lecorvé).

3.3 Systèmes distribués et communication par envoi message [3]

Nous allons maintenant présenter la communication interne à un système distribué et les envois de message.

Présentons dans un premier temps les envois de message dans un système distribué.

D'un point de vue technique, un message est un envoi de données sur un canal. Un message peut, par exemple, être un appel ou un retour de fonction. Pour un système distribué, l'approche la plus courante concernant l'envoi de message est la communication par MPI (Message Passing Interface). Dans un réseaux à deux points, on utilise les sockets qui consiste à envoyer des messages via un canal bidirectionnel mis en place entre les deux points du réseaux. Le protocoles TCP est un exemple de socket. Les MPI peuvent être vu comme la généralisation des sockets dans le domaines de l'algorithmique répartie. L'avantage de la communication par MPI est que cette méthode de communication est générique et son adaptation à chaque architecture ne nécessite que peu de changements. De plus, cette méthode utilise les fonctions présente dans le système d'exploitation utilisé. Avec les MPI, l'envoi et la reception de message entre les composants d'un système distribué est géré ainsi que les connections internes au système.

3.4 Orchestration et chorégraphie des systèmes distribués [13]

L'orchestration d'un système est un processus automatique d'organisation, de coordination ou de gestion. On parle souvent de ce processus comme ayant une intelligence propre ou un contrôle sur le système. Cependant, il s'agit plutôt d'analogies plutôt que de descriptions techniques. En réalité, l'orchestration est la conséquence de l'automatisation d'un système déployant un processus de régulation.

La chorégraphie est une généralisation de l'orchestration. Cela consiste à concevoir une coordination décentralisée des clients. Il n'y a pas de machines privilégiées par le serveur du service utilisé mais le système distribué utilise ses différents composants afin d'effectuer les différents calculs.

La chorégraphie est la base de la technologie des services web. En effet, l'approche que propose la chorégraphie permet de mettre en place le traitement des erreurs et l'approche dynamique des calculs sur les systèmes distribués. La mise en place de cette méthode permet de formaliser les services proposés par le système distribués utilisé.

4 L'approche correct-by-construction [4],[5],[6],[7],[8],[9],[10]

Maintenant que nous avons présenté les outils utilisés, présentons l'approche correct-by-construction.

4.1 Présentation [17]

Le développement logiciel s'appuie canoniquement sur le cahier des charges. Ce cahier est écrit en language naturel mais peut laisser des zones d'ombre. Certaines notations graphiques à l'instart d'UML, permettent de modéliser certains comportements.

Malgré cette possibilité, tous les comportements logiciels ne sont pas modélisables. Une approche correct-by-construction considère le développement logiciel comme un problème d'ingénierie classique. Tout comme l'élaboration de produit industriel "standard", cette approche commence par définir le modèle utilisé. Dans le cas de l'informatique, le modèle en question est un modèle mathématique. Ensuite, le modèle permet de déduire les fonctionnalités du système ainsi que son comportement.

Au contraire de certaines approches qui consistent à déduire les comportements à partir des fonctionnalités, les approches correct-by-construction déduise les comportements et les fonctionnalités du modèle de départ.

4.2 Gain sur les systèmes distribués étudiés

Maintenant que nous avons présenté l'approche correct-by-construction, nous allons étudier les conséquences d'une telle approche sur les systèmes distribués.

Il existe deux types de communication dans les systèmes distribués étudiés. La communication synchrone consiste à avoir une communication sans temps d'attente entre les différents composants du système.Un exemple de communication synchrone est une visio-conférence. Les différents participants ne sont pas tous au même endroit et la communication se fait sans temps d'attente pour l'envoi de message. La communication asynchrone consite à considérer que les différents du système ne communique pas à la même vitesse. Par exemple, lors une conversation orale, les deux protagonistes parlent à la même vitesse : la communication est donc synchrone. Si cette conversation a lieu par mail, la communication est alors asynchrone.

Dans le cas de la communication asynchrone, la méthode de preuve par CP (Conversation protocole) permet de créer des systèmes distribués prouvés. Cette manière de créer permet d'obtenir des systèmes plus robustes et plus fiables grâce à l'utilisation des systèmes état-transition. De plus, l'instanciation proposé par la même équipe de chercheurs permet de mettre en place des systèmes distribués qui peuvent avoir n'importe quel nombre de composants. Les différentes méthodes de création de tels systèmes permettent de créer des systèmes robustes et fiables. La méthode de communication du système n'influe pas sur la création, la méthode de création étant générique.

4.3 L'approche correct-by-construction et les systèmes Web

Présentons les conséquences de l'approches correct-by-construction lorsqu'elle est utilisée sur un système de service web.

Commençons par rappeler l'architecture des ces systèmes. Les systèmes web sont basés sur la communication. Il existe plusieurs couches dans une architecture web classique. La communication entre les clients et le serveur permet de mettre en place des processus de description et d'exécution.

Pour mettre en place de tels systèmes, afin de respecter les normes éditées dans le web (sur le site de la w3c), des langages ont été mis en place.

L'utilisation de la méthode correct-by-construction sur les services web permet de vérifier la complétude des modèles utilisés et de garantir la stabilité des systèmes étudiés. Les différents services des systèmes web sont considérés comme des étapes de raffinement pour la méthode Event-B. Ainsi, plus le nombre de raffinements est élevé, plus le nombre de services prouvés est important.

5 Conclusion

Nous avons présenté les différentes approches qui vont être utilisées dans le cadre de ce stage sur la preuve formelle et la vérification de systèmes critiques. L'utilisation de la méthode Event-B permet de construire de manière incrémentale les sytèmes souhaités tout en garantissant leur bon fonctionnement via l'approche correct-by-construction.

Références

[1] *The B-Tools* Jean-Raymond Abrial 1988

[2] *Communicating sequential processes*, Communications of the ACM, vol 21, no 8, p. 666–677 Charles Antony Richard Hoare, 1978

[3] http://mpi-forum.org/docs/

[4] *Modélisation et vérification formelles de compositions de services. Une approche fondée sur le raffinement et la preuve* Idir AIT-SADOUNE, 2010

[5] *Formal verification of runtime compensation of web service composition : A refinnement and proof-based proposal with Event-B* dans Services Computing (SCC), 2015, IEEE International Conference de Juin 2015, pp. 98-105, Guillaume Babin, Yamine Ait-Ameur, Marc Pantel

[6] *A correct-by-construction model for asynchronously communicationg systems* Zoubeyr Farah, Yamine Ait-Ameur, Meriem Ouederni, Kamel Tari, 2016

[7] *E.W.: A Discipline of Programming*, 1st edn. Prentice Hall PTR, Upper Saddle River,Dijkstra, 1977

[8] *Correct instantiation of a system reconfiguration pattern: a proof and refinement-based approach*, 17th IEEE International Symposium on High Assurance Systems Engineering, HASE 2016, Orlando, FL, USA, January 7–9, pp. 31–38. IEEE Computer Society Press, New York, Guillaume Babin, Yamine Ait-Ameur, Marc Pantel,2016

[9] *A formal Model for Plastic human Interface* K Chebib, Y Ait-Ameur Dans Fontiers of Computer Science, 2017

[10] *Correct-by-construction Evolution of realisable conversation Protocols*, p260-273, S. Benyagoub, M. Ouederni, N. K. Singh, Y. Aït Ameur, 2016

[11] *Algorithmes distribués & Protocoles*, M. Raynal, Eyrolles, 1985

[12] *Une introduction à l'algorithmique distribuée des systèmes asynchrones*, M. Raynal, 1992

[13] *Web Services Orchestration and Choreography*, Peltz, ComputerJournal, Vol. 36(No. 10):46–52. IEEE Computer Society Press, 2003.

[14] *Langages formels, calculabilité et complexité*, Olivier Carton, Paris, Vuibert, coll, "Capes-agrég", 28 octobre 2008, 1e éd., 240 p., 17 x 24 (ISBN 978-2-7117-2077-4 et 2-7117-2077-2)

[15] http://www.methode-b.com/preuve-formelle/

[16] https://fr.wikipedia.org/wiki/Méthode_formelle_(informatique)

[17] http://www.eschertech.com/products/correct_by_construction.php