HDR

Habilitation à Diriger des Recherches

Délivré par : *l'Institut National Polytechnique de Toulouse (Toulouse INP)*

Présentée et soutenue le 06/07/2018 par :

AURÉLIE HURAULT

Formalisations pour les compositions de services

JURY

BERNADETTE CHARRON-BOST Directrice de recherche – CNRS LIX	Rapporteur
CATHERINE DUBOIS Professeur des universités – ENSIIE	Rapporteur
PASCAL POIZAT Professeur des universités – Université Paris Nanterre	Rapporteur
SANDRINE BLAZY Professeur des universités – Université Rennes 1	Examinatrice
BERTRAND MEYER Professeur des universités – Politecnico di Milano et Innopolis University	Examinateur
MARC SHAPIRO Directeur de recherche – INRIA	Examinateur
PHILIPPE QUÉINNEC Professeur des universités – Université de Toulouse	Correspondant

Unité de Recherche :

Institut de Recherche en Informatique de Toulouse (UMR 5505)

Résumé

L'omniprésence de l'informatique dans notre quotidien entraîne un besoin croissant de garanties sur les programmes et systèmes informatiques. Je défends une approche par modélisation et vérification formelles, où les résultats sont mécanisés pour les rendre accessibles à des non-experts des méthodes formelles. Mon domaine d'application est la composition de services. En effet, plusieurs facettes de l'informatique engendrent des problématiques similaires : la programmation orientée objets, la programmation orientée composants, la programmation orientée services, les systèmes répartis ont tous pour but de composer les objets / composants / services / processus pour modéliser et réaliser un système complexe. Mes travaux abordent, sous un angle formel, la composition de services au sens large et les problèmes de découverte, sélection, vérification et adaptation qui en découlent. Plusieurs points de vue sur les services sont abordés : la fonctionnalité qu'ils réalisent, leur comportement (interactions) et leur qualité de service. Certains travaux s'intéressent à la découverte de compositions de services guidée par la fonctionnalité souhaitée. Les techniques utilisées sont celles issues des types abstraits algébriques et de l'unification équationnelle. D'autres travaux s'intéressent à la sélection de la meilleure composition de service en fonction de critères de qualité de service. L'approche se base sur des algorithmes d'optimisation multi-critères et d'apprentissage automatique. Enfin, des travaux s'intéressent à vérifier des propriétés sur des compositions de services interagissant par messages de façon asynchrone. La diversité de la communication dans le monde asynchrone a été étudiée et formalisée.

Table des Matières

CHAPITRE 1

CURRICULUM VITAE

1.1 Parcours

1.1.1 Parcours professionnel

- **Depuis nov 2007**
  Maître de conférences à Toulouse INP - IRIT - Toulouse
- **15 mars 2007 - oct 2007**
  Stage postdoctoral à National Institute of Informatics, Tokyo, Japon
- **sept. 2006 - 15 mars 2007**
  Demi ATER à Toulouse INP - IRIT - Toulouse
- **sept. 2003 - août 2006**
  Allocatrice MENRT / monitrice à Toulouse INP - IRIT - Toulouse

1.1.2 Parcours universitaire

- **déc. 2006**
  Doctorat en informatique de Toulouse INP - IRIT - Toulouse
  Titre : "Courtage sémantique de services de calcul"
  Directeurs de thèse : Michel Daydé et Marc Pantel
- **sept. 2003**
  Diplôme d'Etudes Approfondies - formation doctorale : Programmation et Système de Toulouse INP - IRIT - Toulouse

- **juin 2003**
  Diplôme d'ingénieur en Informatique et Mathématiques appliquées de ENSEEIHT - Toulouse

1.2 Production Scientifique

Résumé de la production scientifique	
Revues internationales	7
Conférences internationales	12
Revues nationales	2
Conférences nationales	2
Autres	2
TOTAL	**25**

1.2.1 Revues internationales

[1] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. *Formal Aspects of Computing*, 28(5) :847–879, 2016.

[2] Serial Rayene Boussalia, Allaoua Chaoui, Aurélie Hurault, Meriem Ouederni, and Philippe Quéinnec. Multi-objective quantum inspired cuckoo search algorithm and multi-objective bat inspired algorithm for the web service composition problem. *IJISTA*, 15(2) :95–126, 2016.

[3] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. Automated verification of asynchronous communicating systems with TLA+. *ECEASST*, 72 : Special issue of the 15th International Workshop on Automated Verification of Critical Systems (AVoCS 2015) :1–15, 2015.

[4] Aurélie Hurault, Kyungim Baek, and Henri Casanova. Selecting linear algebra kernel composition using response time prediction. *Software : Practice and Experience*, 45(12) :1659–1676, 2015.

[5] Hrachya V. Astsatryan, Vladimir Sahakyan, Yuri Shoukouryan, Michel J. Daydé, Aurélie Hurault, Ronan Guivarch, Harutyun Terzyan, and Levon Hovhannisyan. On the easy use of scientific computing services for large scale linear algebra and parallel decision making with the p-grade portal. *J. Grid Comput.*, 11(2) :239–248, 2013.

[6] Yinan Li, Asim YarKhan, Jack J. Dongarra, Keith Seymour, and Aurélie Hurault. Enabling workflows in gridsolve : request sequencing and service trading. *The Journal of Supercomputing*, 64(3) :1133–1152, 2013.

[7] Aurélie Hurault, Michel J. Daydé, and Marc Pantel. Advanced service trading for scientific computing over the grid. *The Journal of Supercomputing*, 49(1) :64–83, 2009.

1.2.2 Conférences internationales

[8] Adam Shimi, Aurélie Hurault, and Philippe Quéinnec. Asynchronous Message Orderings Beyond Causality (regular paper). In *International Conference On Principles Of Distributed Systems (OPODIS), Lisboa, Portugal, 18/12/2017-20/12/2017*, 2017.

[9] Nathanaël Sensfelder, Aurélie Hurault, and Philippe Quéinnec. Inference of channel priorities for asynchronous communication. In *Distributed Computing and Artificial Intelligence, 14th International Conference, DCAI 2017, Porto, Portugal, 21-23 June, 2017*, volume 620 of *Advances in Intelligent Systems and Computing*, pages 262–269. Springer, 2017.

[10] Florent Chevrou, Aurélie Hurault, Philippe Mauran, and Philippe Quéinnec. Mechanized refinement of communication models with TLA$^+$. In *5th Intl. Conf. Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2016)*, pages 312–318. Springer-Verlag, May 2016.

[11] Serial Rayene Boussalia, Allaoua Chaoui, and Aurélie Hurault. Qos-based web services composition optimization with an extended bat inspired algorithm. In *Information and Software Technologies - 21st International Conference, ICIST 2015, Druskininkai, Lithuania, October 15-16, 2015, Proceedings*, volume 538 of *Communications in Computer and Information Science*, pages 306–319. Springer, 2015.

[12] Frédéric Camillo, Eddy Caron, Ronan Guivarch, Aurélie Hurault, Christian Klein, and Christian Pérez. Resource Management Architecture for Fair Scheduling of Optional Computations (regular paper). In *3PGCIC - Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Compiegne, 28/10/2014-30/10/2014*, pages 113–120. IEEE, 2013.

[13] Hrachya Astsatryan, Vladimir Sahakyan, Youri Shoukouryan, Michel Daydé, and Aurélie Hurault. Enabling Large-Scale Linear Systems of Equations on Hybrid HPC Infrastructures (regular paper). In *ICT Innovations 2011, Skopje, 14/09/2011-16/09/2011*, volume 150 of *Advances in Intelligent and Soft Computing*, pages 239–245. Springer, 2012.

[14] Hrachya Astsatryan, Vladimir Sahakyan, Youri Shoukouryan, Myasnik Srapyan, Michel Daydé, Aurélie Hurault, and Romulus Grigoras. Introduction of a Grid-Aware Portlet for Numerical Calculations (regular paper). In *International Conference On Parallel, Distributed and Grid Computing (PDGC), Jaypee University of Information Technology Waknaghat, Solan, H.P., India, 28/10/2010-30/10/2010*, pages 67–70, http ://ieeexplore.ieee.org/, janvier 2011. IEEExplore digital library.

[15] Aurélie Hurault and Asim YarKhan. Intelligent service trading and brokering for distributed network services in gridsolve. In *High Performance Computing for Computational Science - VECPAR 2010 - 9th International conference, Berkeley, CA, USA, June 22-25, 2010, Revised Selected Papers*, volume 6449, pages 340–351. Springer, 2010.

[16] Hrachya V. Astsatryan, Vladimir Sahakyan, Yuri Shoukouryan, Michel J. Daydé, Aurélie Hurault, Marc Pantel, and Eddy Caron. A grid-aware web portal with advanced service trading for linear algebra calculations. In *High Performance Computing for Computational Science - VECPAR 2008, 8th International Conference, Toulouse, France, June 24-27, 2008. Revised Selected Papers*, volume 5336, pages 150–159. Springer, 2008.

[17] Michel J. Daydé, Aurélie Hurault, and Marc Pantel. Semantic-based service trading : Application to linear algebra. In *High Performance Computing for Computational Science - VECPAR 2006, 7th International Conference, Rio de Janeiro, Brazil, June 10-13, 2006, Revised Selected and Invited Papers*, volume 4395, pages 622–633. Springer, 2006.

[18] Aurélie Hurault and Marc Pantel. Mathematical Service Trading Based on Equational Matching. In *Calculemus'05 : 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Newcastle, 18/07/2005-22/07/2005*. Electronic Notes in Theoretical Computer Science, juillet 2005.

[19] Michel Daydé, Aurélie Hurault, and Marc Pantel. Gridification of Scientific Application Using Software Components : the Grid-TLSE Project as an illustration. In *CSIT 2005, Yerevan, Armenia, 19/09/2005-23/09/2005*, pages 419–427, Yerevan, Armenia, septembre 2005. National Academy of Siences of Armenia.

1.2.3 Revues nationales

[20] Aurélie Hurault, Marc Pantel, and Michel J. Daydé. Composition dynamique de services de calcul. utilisation des spécifications algébriques pour décrire la fonctionnalité des services. *Technique et Science Informatiques*, 30(6) :685–710, 2011.

[21] Aurélie Hurault, Vincent Hennebert, and Marc Pantel. Répartition et mobilité en JAVACT une approche dérivée d'un modèle formel. *L'OBJET*, 10(2-3) :47–60, 2004.

1.2.4 Conférences nationales

[22] Florent Chevrou, Aurélie Hurault, Philippe Mauran, Meriem Ouederni, Philippe Quéinnec, and Xavier Thirioux. La composition de services dans le monde asynchrone Formalisation et vérification en TLA+ (short paper). In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL), Bordeaux, 09/06/15-10/06/15*, pages 34–39, http ://gdr-gpl.cnrs.fr, juin 2015. CNRS - GDR GPL.

[23] Aurélie Hurault, Marc Pantel, and Frédéric Desprez. Recherche de services en algèbre linéaire sur une grille. In *Rencontre du Parallélisme, Le Croizic, France, 06/04/2005-08/04/2005*, pages 207–212. EMN-ASF, avril 2005.

1.2.5 Autres contributions significatives

[24] *AFADL'16, Actes des 15èmes journèes sur les Approches Formelles dans l'Assistance au*

Developpement de Logiciels, Besançon, France, 2016. Aurélie Hurault et Nicolas Stouls, editeurs.

[25] Aurélie Hurault. *Courtage sémantique de services de calcul*. Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, décembre 2006. (Soutenance le 04/12/2006).

1.3 Encadrements

1.3.1 Encadrement de doctorats

Adam Shimi

- **Sujet** : Vérification de systèmes distribués paramétrés avec défaillances.
- **Date de début de Thèse** : octobre 2017
- **Liste des travaux co-publiés**
  - une conférence internationale [SHQ17]
- **Co-encadrement avec** : Philippe Quéinnec
- **Financement** : Bourse de thèse de l'ANR

Florent Chevrou

- **Sujet** : Formalisation des interactions asynchrones.
- **Date de début de Thèse** : octobre 2014
- **Date de soutenance** : 22 novembre 2017
- **Liste des travaux co-publiés**
  - deux revues internationales [CHQ15, CHQ16]
  - une conférence internationale [CHMQ16]
  - une conférence nationale [CHM+15]
- **Co-encadrement avec** : Philippe Quéinnec
- **Financement** : Bourse du ministère
- **Situation actuelle** : ATER

Seriel Boussalia

- **Sujet** : Utilisation des méthodes d'optimisation pour trouver une meilleure solution de composition de services web
- **Laboratoire** : Laboratoire MISC Laboratory, université de Constantine 2, Algérie
- **Date de début de Thèse** : 2012

- **Date de soutenance** : 25 mai 2016
- **Liste des travaux co-publiés**
  - une revue internationale [BCH+16]
  - une conférence internationale [BCH15]
- **Co-encadrement avec** : Allaoua Chaoui
- **Situation actuelle** : Maître de conférences à l'université de Constantine 1

1.3.2 Encadrement de masters

Nom, prénom de l'étudiant	Sujet	Période
Adam Shimi	Impact de la paramétrisation dans les compositions de services	01/03/2017 - 01/09/2017
Nathanael Sensfelder	Modèles de communication applicatifs pour la composition de services répartis	01/03/2016 - 01/09/2016
Florent Chevrou	Modélisation et vérification de compositions de services répartis	01/03/2014 - 01/09/2014

1.4 Responsabilités collectives

1.4.1 Animation du groupe MFDL du GDR GPL

Je suis co-animatrice du groupe MFDL (Méthodes Formelles dans le Développement Logiciel) du GDR GPL depuis janvier 2016. http://gdr-gpl.cnrs.fr/Groupes/MFDL/Description

Le groupe MFDL regroupe 22 équipes réparties sur toute la France. Chaque année des présentations de membres du groupe sont organisées lors des journées du GDR-GPL en collaboration avec la conférence AFADL. Des journées propres au groupe ou en collaboration avec d'autres groupes de travail sont aussi organisées, comme le 7 décembre 2017 : http://lig-membres.imag.fr/idani/MTV2_MFDL/index.html

1.4.2 Responsabilités d'enseignement

Responsable de l'UE de Programmation fonctionnelle J'étais responsable de l'UE de programmation fonctionnelle du (feu) département informatique et mathématiques appliquées de l'ENSEEIHT. Cette UE était en première année d'école d'ingénieur (niveau L3) et correspondait à 60 h de présence étudiante, 100 étudiants, 8 enseignants, 1 projet, 1 bureau d'étude et 2 examens.

Responsable de l'UE de Programmation fonctionnelle et compilation Je suis responsable de l'UE de programmation fonctionnelle et compilation du nouveau département sciences

du numérique. Cette UE est en deuxième année d'école d'ingénieur (niveau M1) et correspond à 60 h de présence étudiante, 70 étudiants, 6 enseignants, 1 projet, 1 bureau d'étude et 2 examens.

Responsabilités de cours en petite promotion Je gère seule mes cours avec les apprentis (15 étudiants en moyenne) de la formation Informatique et Réseaux de l'ENSEEIHT, avec les élèves ingénieurs de l'ENAC (30 étudiants en moyenne) ainsi que les élèves ingénieurs de l'École Nationale de la Météorologie (15 étudiants en moyenne).

1.4.3 Participation à des conseils d'unités ou d'établissement

Coordinatrice des Relations Internationales du département informatique et mathématiques appliquées de l'ENSEEIHT De septembre 2010 à septembre 2017, j'ai été responsable des relations internationales pour le département informatique et mathématiques appliquées de l'ENSEEIHT. Cela consiste, entre autres, à entretenir les contacts avec les universités partenaires et à développer de nouveaux accords en vérifiant l'adéquation des formations.

Cela représente une centaine d'heures et plus de mille mails d'étudiants et de collègues à traiter par an.

Membre du conseil de formation de la formation Informatique et Réseau par la voie de l'apprentissage En septembre 2009, l'ENSEEIHT a ouvert une formation par apprentissage Informatique et Réseaux. Je fais partie du groupe (de 8 personnes) qui a monté la formation et depuis je suis dans le conseil de formation (une réunion par semaine les premières années, une réunion par mois depuis que la formation a atteint son régime de croisière).

J'interviens dans la formation en y donnant des cours et en suivant deux ou trois apprentis par an. Je les accompagne pendant les trois années de la formation en faisant le lien entre l'école et l'entreprise dans laquelle ils réalisent leur apprentissage. Je me rends deux fois par an dans leur entreprise et nous nous voyons une fois par an à l'école pour des demi-journées de présentation des travaux des étudiants.

Membre du conseil de département Informatique et Mathématiques Appliquées de l'ENSEEIHT Depuis mon recrutement, j'ai été membre du conseil de département du département Informatique et Mathématiques Appliquées de l'ENSEEIHT. Je viens d'être élue membre du conseil de département du nouveau département Sciences du Numérique.

Membre du vivier section 27 Depuis début 2013, je suis membre du vivier de section 27 (à l'origine de la constitution des comités de sélection). J'ai participé au comité de sélection d'un poste de maître de conférences en 2015 (poste MCF 0188).

1.5 Collaborations internationales

NII : National Institute of Informatics, Tokyo, Japon Suite à mon séjour de 8 mois au NII (15 mars 2006 - 31 octobre 2007) lors de mon stage postdoctoral, des collaborations ont perdurées. Par exemple, Florent Chevrou, un doctorant que j'ai co-encadré, y a fait un séjour de 2 mois en 2015. Les travaux réalisés sont présentés dans la section 3.2.3.

Institute for Informatics and Automation Problems of NAS RA, Yerevan, Arménie Depuis le début de ma thèse, j'entretiens une collaboration avec l'Institute for Informatics and Automation Problems, National Academy of Sciences of the Republic of Armenia. Cette collaboration a donné lieu à de nombreux séjours et visites dans les deux sens. Les travaux réalisés sont présentés dans la section 3.2.2.

ICL : Innovative Computing Laboratory, Université du Tennessee, USA En 2009 j'ai effectué un séjour de trois mois à l'université du Tennessee dans le laboratoire ICL. Ce séjour a été financé par mon institution (Toulouse INP) et mon équipe (ACADIE). Les travaux réalisés sont présentés dans la section 3.2.1.

Information and Computer Sciences Department, Université d'Hawaï à Manoa, USA Début 2012, j'ai effectué un séjour de 2 mois à l'université de Hawaii à Manoa, pour une collaboration avec Henri Casanova. Ce séjour a été financé par mon institution (Toulouse INP) et mon équipe (ACADIE). Les travaux réalisés sont présentés dans la section 5.1.

MISC laboritary, université de Constantine 2, Algérie Ces dernières années, j'ai également co-encadré Seriel Boussalia, en collaboration avec le laboratoire MISC (université de Constantine 2, Algérie). Seriel a réalisé deux séjours de quelques mois à Toulouse, et j'ai réalisé une visite à l'université de Constantine 2. Les travaux réalisés sont présentés dans la section 5.2.

1.6 Participation / coordination de projets

Pardi

- Rôle : Membre
- Sujet : Vérification de systèmes distribués paramétrés
- Financement : ANR-16-CE25-0006
- Partenaires :
    - Toulouse INP / IRIT – porteur
    - Université Paris sud / LRI

    - INRIA Nancy - Grand Est
    - Université Pierre et Marie Curie / LIP6
- Période : 2016 - 2020
- Budget : 504 k€(total) - 155 k€(Toulouse INP)
- http://pardi.enseeiht.fr

SNOB

- Rôle : Porteur du projet
- Sujet : preuves mécanisées de Systèmes NOn Bloquants
- Financement : Toulouse Tech InterLabs
- Partenaires : équipe ACADIE de l'IRIT et équipe TSF du LAAS
- Période : 01/06/2017 - 31/12/2018
- Budget : 5 000 €

Cocasse

- Rôle : Porteur du projet
- Sujet : Composition certifiée de services basée sur la QoS
- Financement : Toulouse Tech InterLabs
- Partenaires : équipe ACADIE de l'IRIT et équipe SARA du LAAS
- Période : 01/06/2014 - 31/12/2015
- Budget : 9 000 €

COOP

- Rôle : Membre
- Sujet : Gestion de ressources coopérative Multi-niveaux
- Financement : ANR Cosinus 2009
- Partenaires :
    - INRIA Lyon (équipe GRAAL)
    - INRIA Rennes (projet Myriads)
    - INRIA Bordeaux (groupe Runtime)
    - Toulouse INP (équipes APO et ACADIE)
    - EDF R&D / SINETICS.
- Période : 01/12/2009 - 31/05/2013
- Budget : 110 000 €
- http://coop.gforge.inria.fr/doku.php/start

GRID-TLSE

- Rôle : Co-responsable pour l'IRIT à partir de mon recrutement
- Sujet : Site d'expertise en algèbre linéaire creuse
- Financements successifs :
  - ACI "Globalisation des Ressources Infomatiques et des Données"
  - Projet ReDIMSoPS via la coopération CNRS/JST (Japon)
  - Projet SOLSTICE (ANR-06-CIS6-010)
  - Projet LEGO (ANR-CICG05-11)
  - Projet FP3C, projet collaboratif entre la France et le Japon (ANR-JST FP3C)
  - Projet COOP (ANR-09-COSI-001)
- Partenaires initiaux :
  - les partenaires académiques : CERFACS, IRIT, LaBRI et LIP-ENS Lyon
  - les partenaires industriels : CEA, CNES, EDF et IFP
- Période : 01/11/2002 - 31/05/2013

CHAPITRE 2

CONTEXTE

La composition de services est un vaste domaine qui a pris de l'ampleur avec les années. Souvent restreinte à tort à la composition de services web [ACKM10], elle va être ici abordée dans un contexte plus large. En effet, plusieurs facettes, à priori sans lien, de l'informatique engendrent des problématiques identiques relatives à la composition de services. Citons par exemple les paradigmes de programmation mettant en avant une approche modulaire de la programmation tels que la programmation orientée objets [MJ00], la programmation orientée composants [Hei01] ou encore la programmation orientée services [Bel08]. Le but ultime est de pouvoir réutiliser des objets / composants / services comme des boîtes noires et de les composer entre eux pour obtenir une application plus complexe. Citons également les systèmes répartis / distribués où, par nature, des codes distants / isolés doivent se composer pour atteindre un but commun.

Un des objectifs des paradigmes de programmation modulaire est la mutualisation / réutilisation de code. Il y a en effet des avantages à cela comme le gain de temps en ne redéveloppant pas du code existant ou le gain en fiabilité du code (plus il sera utilisé, plus les bugs seront détectés et corrigés). Les paradigmes de programmation orientée objets, services ou composants, et même l'utilisation de bibliothèques existantes, permettant d'écrire un logiciel sans avoir à écrire tout le code, mais en assemblant des morceaux déjà écrits. C'est cet assemblage que j'appellerai "composition de services". Malgré le développement de ces approches, aujourd'hui encore beaucoup de code neuf est produit quotidiennement, beaucoup de code est redéveloppé alors que du code réalisant le même objectif existe déjà. Ce serait mentir que de dire que la belle idée du "composant sur étagère" a vraiment pris. Une des raisons est sans doute la difficulté à réutiliser et composer les modules / services. Comment être sûr que le service réalise bien la fonctionnalité souhaitée ? Que se passe-t-il lorsque plusieurs services sont combinés ?

Le cloud computing [BBG11, AG10] ou l'internet des objets [Gre15] sont des mots à la mode

dont certaines problématiques rejoignent celles déjà connues et abordées depuis des années dans les systèmes répartis / distribués. La multiplication des "ordinateurs" (au sens large : téléphones, objets connectés, calculateurs de voiture...), augmente le besoin de faire interagir des services distants. Il s'agit par nature d'une problématique de composition de services (plutôt appelés pairs dans le monde des systèmes distribués). En plus des questions précédentes sur les fonctionnalités rendues par les services, s'ajoutent des problèmes liés à la répartition et à la communication. Est ce que deux services distants arrivent à communiquer entre eux ? A quelles conditions ? S'ajoute également des problématiques de qualité de services quand on aborde le cloud computing par exemple. En effet l'utilisation des plateformes ou des services pouvant être payante, les questions de coût ou encore de temps d'exécution deviennent importantes.

Une dernière évolution de l'informatique ayant été prise en compte dans ces travaux est la criticité de l'informatique. Si dans certains domaines, il est obligatoire de respecter des normes de codage (dans l'espoir de faire disparaître tout bug), c'est loin d'être le cas de tous les domaines. Pour les domaines critiques, les méthodes formelles ont été développées et utilisées. Néanmoins elles restent confidentielles à ces domaines critiques. Il est encore trop banal pour les utilisateurs que leur ordinateur ou téléphone n'ait pas le comportement escompté. Avec la multiplication des ordinateurs dans notre quotidien, ces petites gênes peuvent se transformer en situation critique. L'utilisation et la généralisation des méthodes formelles est une réponse à cette problématique. Leurs difficultés d'utilisation ralentissent leur diffusion mais les énormes progrès de ces dernières années, notamment en matière de preuve grâce aux solveurs SMT, donnent de l'espoir.

Dans ces travaux, les problématiques sont plutôt abordées formellement et avec un point de vue théorique, il y a néanmoins toujours une implantation "preuve de concepts" réalisée. En effet, il est important que les résultats théoriques puissent être utilisés en pratique. Ces implantations étant issues de formalisations, elles ont, dans la mesure du possible, été prouvées correctes et complètes par rapport aux spécifications. Le but n'a jamais été une industrialisation des développements, mais de pouvoir vérifier que les idées / concepts théoriques étaient effectivement mécanisables. Cela permet également de mettre en évidence d'éventuels points faibles des approches : expressivité trop faible, problèmes de passage à l'échelle... entraînant une incapacité à traiter des exemples significatifs. Se met alors en place un mécanisme d'aller-retour entre la théorie et la mécanisation : chaque nouveau concept est implanté et chaque limite remontée par la mécanisation est étudiée pour étendre / améliorer la théorie. La combinaison des deux points de vue permet de s'assurer que la théorie est exploitable, et d'avoir une garantie sur le comportement de la mécanisation.

Dans ce manuscrit les problématiques liées à la **composition de services** sont (presque) toujours abordées à l'aide de **méthodes formelles** avec une volonté de **mise en œuvre informatique**.

2.1 Spécification des services

Pour pouvoir réutiliser et assembler des services, il est nécessaire de posséder une spécification de ceux-ci. En effet, sans information sur leur fonctionnalité (que font-ils ?) ou leur comportement (avec quelles interactions avec l'extérieur ?), il est impossible de les utiliser et encore moins de s'en servir comme brique de base pour concevoir une application plus complexe. Usuellement, il existe plusieurs niveaux de spécification des services :

- signature (nom et types) ;
- sémantique — spécification fonctionnelle ;
- comportement — protocole d'interaction ;
- qualité de service – propriétés non fonctionnelles.

2.1.1 La signature

La signature est l'information minimale requise pour utiliser un service et est usuellement combinée avec une des autres approches. Elle est composée du nom, des types des entrées et des types des sorties du service.

La plupart des langages de programmation propose de définir les signatures des méthodes dans une interface. Citons, les fichiers .h de C, les interfaces de Java ou encore les fichiers .mli d'OCaml. En Corba [1], par exemple, les interfaces sont exprimées en IDL [2] (Interface Definition Language) et des mécanismes d'introspection d'interfaces (DII et DSI [GGM99]) permettent la découverte des signatures des services disponibles.

Pour les services web, cette information est disponible dans la description WSDL [3] qui est basée sur XML. Pour chaque opération que propose le service web, son nom, ses paramètres et le type de retour sont spécifiés. Pour chaque paramètre, son type est spécifié. D'autres informations, non fonctionnelles, sont également décrites dans le fichier WSDL : le protocole de communication qui doit être utilisé, le format des messages nécessaires ainsi que la localisation du service.

Il est évident que la signature d'un service est indispensable à sa spécification, mais seule, elle est insuffisante. En effet la signature ne donne aucune information sur la fonctionnalité réalisée par le service. Deux services avec la même signature peuvent résoudre des problèmes complètement différents. Néanmoins, la signature a l'avantage d'être une spécification utilisable facilement par des outils automatiques (les compilateurs savent par exemple très bien les analyser), ce qui, dans une démarche de mécanisation, est un atout majeur.

1. http://www.omg.org/corba/
2. http://www.omg.org/gettingstarted/omg_idl.htm
3. https://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/

2.1.2 La sémantique

En plus de la signature d'un service, il est indispensable de connaître la spécification de sa fonctionnalité (appelée ici sémantique du service). Celle-ci précise le problème résolu par le service : inversion de matrice (par exemple pour l'utilisation de bibliothèque mathématique), réservation d'hôtel (par exemple pour un service web), résolution du consensus (par exemple dans le monde des systèmes distribués)...

Dans de nombreux paradigmes de programmation, les services sont décrits en langage naturel à l'aide de quelques phrases. L'ambiguïté du langage naturel n'est plus à démontrer, la spécification peut alors être non complète, contradictoire, difficilement compréhensible... Il faut ajouter à cela la difficulté du choix de la langue. Cette spécification informelle n'est nullement adaptée à une automatisation des traitements sur les spécifications des services.

Une autre spécification répandue est une approche par mot-clés. La description par mots-clés peut prendre différents aspects pour un service donné : un seul mot-clé, une liste de mots-clés, une unique association attribut / mot(s)-clé(s) ou une liste d'associations attribut / mot(s)-clé(s). Le Corba Trading Service [Gro94] (chapitre 16) permet de décrire les services par une interface contenant entre autres une liste de propriétés sous la forme de séquences de paires (nom de l'attribut, valeur). Sur le même modèle, et dans le monde des services web, RDF [4] propose également un standard de description, basé sur XML, des services. La richesse de RDF est d'offrir une grande liberté dans la valeur associée à un attribut. Cela peut être n'importe quel terme XML. L'avantage de cette approche est sa simplicité, néanmoins ils ne permettent pas d'être très précis sur la spécification du service. Il n'est par exemple pas possible de décrire le rôle de chaque paramètre. Il est également difficile de composer ces mots-clés pour déduire la fonctionnalité rendue par une composition de services. Enfin, si le domaine de définition de ces mots-clés est totalement libre, les problèmes de langage sont de nouveau bloquants et il est nécessaire de mettre en place des mécanismes de standardisation des mots-clés utilisés.

Les ontologies de domaine ont été introduites pour répondre à ce problème. Elles permettent de formaliser un domaine. Dans le monde des services web, le standard est OWL [5] [MvH04]. Dans [AvH03], les auteurs citent les cinq principaux objectifs : une syntaxe bien définie, une sémantique bien définie, un support de raisonnement efficace, une expressivité suffisante, et une facilité d'expression. Pour résoudre le problème d'expressivité et de performance du raisonnement, OWL est découpé en 3 sous-langages qui proposent différents compromis entre expressivité et décidabilité. Le plus utilisé est OWL-DL pour Description Logic. Des outils, tels que Protégé [6] [NSD+01], permettent de définir des ontologies de façon relativement intuitive même pour des utilisateurs novices. Des outils, tels que Hermit [7], Racer [8] ou encore FaCT [9],

4. https://www.w3.org/RDF/
5. https://www.w3.org/2001/sw/wiki/OWL
6. http://protege.stanford.edu/
7. http://www.hermit-reasoner.com/
8. http://www.ifis.uni-luebeck.de/~moeller/racer/index.html
9. http://www.cs.man.ac.uk/~horrocks/FaCT/

permettent de raisonner sur ces ontologies. Mais les ontologies ne sont pas restreintes aux services web et à OWL. S'appelle ontologie toute formalisation d'un domaine associé à un mécanisme de raisonnement sur cette formalisation. F-logic [KL89, KLW95] est un autre langage d'ontologie qui mélange logique du premier ordre et programmation orientée objet. Là encore, il existe des outils pour définir et raisonner sur les ontologies comme Ontobroker [10] ou Flora2 [11].

Parmi les spécifications formelles, la logique de Hoare [Hoa69, Hoa83] permet de décrire les services comme des relations entre deux prédicats : une pré-condition et une post-condition. Si le service démarre dans un état qui satisfait la pré-condition et qu'il termine, alors l'état final satisfera la post-condition. Il y a alors une correction totale. Si la terminaison n'est pas exigée, la correction sera partielle. La logique de Hoare est mécanisable sur les programmes impératifs et orientés objets par exemple (Why3 [12] par exemple implémente cette mécanisation).

Pour des problèmes concurrents, les spécifications sont usuellement écrites grâce aux logiques temporelles. Elles ajoutent, par rapport à la logique des prédicats, des opérateurs qui permettent de raisonner sur le temps, comme "une propriété est *toujours* vraie" ou "*un jour* une propriété sera vraie". Il existe différentes logiques temporelles. Les plus connues sont LTL [MP92], pour Linear Temporal Logic, CTL [CE81] ou CTL* [EH86], pour Computation Tree Logic. Alors que LTL propose une notion de temps linéaire, CTL possède un modèle de temps arborescent, c'est-à-dire qu'il peut y avoir plusieurs futurs. Cette logique propose donc des quantificateurs sur les branches temporelles. CTL*, quant à elle, inclut LTL et CTL. Ces logiques ne sont pas les seules logiques existantes. Citons également ATL et ATL* [AHK02] qui s'intéressent aux structures de jeux concurrentes et qui proposent donc des opérateurs sur les stratégies de jeux. Notons que LTL et CTL sont utilisées en particulier pour la possibilité de mécanisation de la vérification de leurs formules grâce à la vérification de modèle (model checking). CTL a d'ailleurs été inventée en même temps que la vérification de modèle [CE81].

La spécification peut également être réalisée en se basant sur la théorie des ensembles. Les formalismes associés sont la méthode B [Abr96], la méthode Event-B [13] [Abr10], le langage Z [Spi89] ou encore VDM [vdm78, FLM+05]. L'avantage de ces spécifications c'est qu'elles bénéficient de toute la puissance et la rigueur de la théorie des ensembles. TLA$^+$ [Lam02] est également un langage de spécifications formelles. Il est basé sur la théorie des ensembles non typés de Zermelo-Fraenkel pour spécifier les structures de données et sur la logique temporelle d'actions (temporal logic of actions (TLA)) pour spécifier les comportements dynamiques. Les expressions s'appuient sur la logique du premier ordre, les opérateurs ensemblistes et un certain nombre de modules arithmétiques.

10. https://www.w3.org/2001/sw/wiki/OntoBroker
11. http://flora.sourceforge.net/
12. http://why3.lri.fr/
13. http://www.event-b.org/

2.1.3 Le comportement

En plus des informations précédentes (signature et sémantique), il peut être intéressant de savoir comment le service interagit avec l'environnement / les autres services. C'est ce qui est appelé ici son comportement. Il s'agit par exemple de spécifier dans quel ordre il envoie / reçoit des messages à destination / en provenance d'autres services. Un service de réservation d'hôtel aura la même sémantique qu'il demande à recevoir la date avant le lieu ou inversement, par contre il ne se composera pas de la même façon avec d'autres services. Dans le monde des systèmes répartis, il est nécessaire de connaître le comportement de chaque service pour vérifier que les services seront capables d'interagir.

Dans le monde des services web, le langage BPMN [14] propose trois niveaux de description. Le premier niveau permet de décrire chaque service par des activités, des évènements, des flux de contrôle, des branchements, des messages et des données. Il permet également de décrire une collaboration, en précisant les échanges entre les différents services. Enfin, il permet de décrire des chorégraphies c'est-à-dire le comportement global, souhaité de la composition de service. La sémantique de BPMN n'a malheureusement pas été définie formellement, il reste donc des ambiguïtés dans les spécifications.

Il existe néanmoins des outils formels pour décrire le comportement des services. Une approche classique est l'utilisation de systèmes de transitions. Ils sont au cœur de méthodes formelles de spécification et vérification tels que TLA$^+$ [Lam78] (Temporal Logic of Actions), Promela [Hol97] (Process Meta Language) ou la méthode Event-B [Abr10]. Les Input/Output automata [Lyn96] fournissent un cadre naturel de description de services qui interagissent entre eux à l'aide d'actions d'entrée et de sortie. Autant les systèmes de transitions bénéficient d'une large couverture d'outils, autant peu d'outils ont été développés pour raisonner sur les I/O automata. Finalement les calculs de processus, tels que CCS (Calculus of Communicating Systems) [Mil82], CSP (Communicating Sequential Processes) [Hoa78], LOTOS (Language of Temporal Ordering Specification) [SCG$^+$00] ou encore le π-calcul [Mil99], permettent également de spécifier un processus / service et ses interactions. Les services sont décrits de façon simple et concise à l'aide d'opérateurs de composition de processus (composition parallèle, séquence, choix déterministe ou pas, ...). Les processus élémentaires représentent des actions, souvent des liées à la communication.

Nous voyons que le choix de formalisation est grand. Néanmoins la question de savoir si les différents services arrivent à communiquer entre eux reste difficile et particulièrement quand nous nous plaçons dans le monde de la communication asynchrone qui est une des problématiques abordées dans ce manuscrit.

14. http://www.omg.org/spec/BPMN/

2.1.4 La qualité de service

La qualité de service, enfin, fait référence à des propriétés non fonctionnelles des services tels que le coût (dans le cas de service payant), le temps d'exécution, la disponibilité... Ces aspects ne sont que très peu abordés dans le manuscrit. Ils sont majoritairement utilisés en vue de la sélection d'une composition de services.

2.1.5 Positionnement

Dans ce manuscrit, les différents niveaux de description sont abordés plus ou moins en détail. La signature étant la base indispensable, elle est présente dans la quasi-totalité des travaux.

Le courtier de services de calcul (voir section 3.1) est basé sur des descriptions proches des types abstraits algébriques. Le domaine des services est défini par un ensemble d'opérateurs et d' équations sur ces opérateurs. Cela peut être vu comme une ontologie (au sens : formalisation d'un domaine d'application). Ce formalisme permet de décrire précisément le rôle de chaque paramètre. Néanmoins cette approche n'est possible que pour des domaines d'applications bien spécifiques, car il n'est pas toujours possible de définir le domaine avec un type abstrait algébrique. Pour des raisons de non maturité à l'époque des travaux, les ontologies n'ont pas été utilisées, mais si les travaux étaient à refaire aujourd'hui, elles seraient à étudier attentivement. Elles ont cependant été utilisées dans des travaux annexes (voir section 3.2.3.2) pour décrire les machines sur lesquelles est déployée la composition de services. Enfin, une description à base de workflow (voir section 3.3) est utilisée pour décrire des scénarios de calculs dans le monde de l'algèbre linéaire creuse.

Les aspects comportementaux ont été abordés en utilisant des systèmes de transition et les méthodes TLA$^+$ et Event-B (voir chapitre 4). La particularité de ces travaux est de se placer dans un monde asynchrone.

Le point de vue de la qualité de service n'est que peu présent dans ces travaux (voir chapitre 5). Les descriptions sont classiquement des valeurs de coût (temps d'exécution par exemple) codées par des valeurs numériques.

2.2 Composition de services

Nous venons de discuter des différents niveaux de description / spécification des services ainsi que différents formalismes dans lesquels il est possible de les réaliser. Nous allons maintenant nous intéresser à la composition de services et aux problématiques qu'elle soulève. Ces problématiques sont de plusieurs natures :

- découverte d'une composition répondant à une spécification ;
- sélection d'une composition parmi plusieurs ;
- vérification qu'une composition répondant à une spécification ;

- adaptation d'une composition quand elle ne répond pas à une spécification.

La composition d'un ensemble de services pouvant être vue elle-même comme un service plus complexe, sa spécification est généralement dans le même formalisme que les services qui la composent, avec éventuellement des informations complémentaires sur les interactions à l'intérieur de la composition.

2.2.1 La découverte

La découverte d'une composition de services consiste à trouver un ensemble de services qui, combinés, permettent de répondre à une spécification.

Afin de s'intéresser à la découverte d'une composition de service, il est souvent nécessaire d'être en mesure de découvrir un service simple. La recherche d'un service dans une bibliothèque à partir de la signature du service n'est pas un domaine de recherche récent [Rit89, RT89, ZW93, ADC96]. Tous les travaux dans ce domaine soulignent l'importance de ne pas se limiter à une correspondance exacte des types. En effet, la fonctionnalité du service est la même si l'ordre des paramètres du service est permuté. Mais il peut également être intéressant de proposer des services dont la signature se rapproche de celle recherchée sans être, pour autant, exactement identique. Dans le cas de systèmes polymorphes, il peut aussi être intéressant de renvoyer des signatures plus générales que la requête. Dans le monde des services web, UDDI [15] est un annuaire qui permet une recherche des services par pages blanches (à partir du fournisseur du service), par pages jaunes (à partir d'une taxinomie des domaines des services) ou par page verte (à partir d'informations techniques sur le service).

La découverte d'un service simple décrit par mot-clés consiste en une comparaison de mots-clés où à une inclusion de liste de mot-clés. Elle est facilement automatisable, ce qui n'est pas toujours le cas pour des descriptions plus poussées. Selon les besoins, les spécifications doivent être identiques (bissimulation) ou le service peut faire plus de choses que la spécification souhaitée (simulation). Il est aussi possible d'envisager un service qui ferait moins de choses que la spécification attendue, nous serions alors dans un mode dégradé. Cette problématique de la comparaison de deux spécifications intervient également dans les problématiques de réparation de composition, quand un service est en échec et qu'il faut le remplacer.

Lorsqu'un service simple ne permet pas de répondre seul au besoin, il est nécessaire de composer plusieurs services. La découverte d'une composition de services peut être guidée par la connaissance d'un plan de composition général de l'application. La structure générale de la composition est identifiée et modélisée par un ensemble de sous-tâches nécessaires à la réalisation de la spécification globale. Il faut alors, pour chaque sous-tâche, trouver le service (unique) qui la réalise. Cette approche est relativement simple, car le nombre de services nécessaires et leur interaction sont prédéfinis. Le problème se ramène donc à savoir si la spécification d'un service est compatible avec la spécification souhaitée pour la sous-tâche. Le

15. http://soapclient.com/uddisearch.html

plan peut également être connu sous la forme d'une chorégraphie, la question est alors de savoir si cette chorégraphie est réalisable [ALW89, AEY01, KP06, BBO12], c'est-à-dire de savoir s'il est possible de construire un système distribué qui communique exactement comme la chorégraphie le spécifie. Le projet VerChor [GPSY16], par exemple, s'intéresse entre autre à cette problématique.

Lorsque le plan de composition n'est pas connu, il faut le découvrir. Plusieurs techniques existent pour résoudre ce problème. Une étude sur l'utilisation des techniques basées sur la planification IA pour la composition de service web a été réalisée dans [Pee05]. En général un problème de planification est caractérisé par un ensemble d'actions qui peuvent être exécutées (nous retrouvons ici une formalisation du domaine des services), un état initial et un état cible. Des techniques basées sur le chaînage, comme par exemple [BBGB16], peuvent aussi être utilisées. Ces techniques de chaînage s'intéressent à faire correspondre les sorties d'un service aux entrées d'un autre, constituant ainsi une chaîne de services.

2.2.2 La sélection

Il existe rarement une unique composition répondant au besoin, il est donc nécessaire de faire une phase de sélection.

Lorsque toutes les compositions répondent au besoin fonctionnel, c'est sur la qualité de service que la composition est sélectionnée. Il faut choisir la "meilleure". La difficulté est alors de définir ce qu'est la meilleure composition. Le critère peut être unique : celle qui aura le temps d'exécution le plus rapide, celle qui coûtera la moins chère, celle qui utilisera le moins de mémoire... Le critère peut également être multiple. Il faut alors faire le choix des pondérations à donner à chaque critère pour les agréger. Dans la littérature, peu de travaux s'intéressent à ce problème. Il est souvent supposé que l'utilisateur a une idée claire des poids à assigner, mais ce n'est pas forcément réaliste. Certains travaux proposent à l'utilisateur plusieurs compositions. Une partie des travaux s'intéressent alors à trouver les compositions de services représentant le front de Pareto.

Il faut également "un modèle de calcul" de la composition en fonction des coûts des services. En effet, à partir de la qualité de service de chaque service, il faut calculer la qualité de service de la composition complète. Pour des caractéristiques numériques (coût et temps d'exécution par exemple), la qualité de service globale peut être la somme des qualités de service (raisonnable pour le coût et pour le temps d'exécution si les exécutions sont séquentielles), la moyenne, le maximum (pour le temps d'exécution de services parallèles), le produit... Dans [JRGM05, ZBN+04] différentes fonctions d'agrégation sont proposées en tenant compte des éléments structurels de la composition de services (chemins parallèles, séquences, boucles...).

D'autres travaux ajoutent d'autres critères de sélection (en plus ou en remplacement de la qualité de service). Par exemple dans [GAJG16], les auteurs s'intéressent à des contraintes temporelles (intervalles de temps dans lequel la tâche doit être réalisée). Dans [YB12] l'ensemble

des services sélectionnés n'est pas le front de Pareto, car est pris en compte la volonté de l'utilisateur de privilégier certains fournisseurs.

2.2.3 La vérification

Un problème complémentaire à la découverte est la vérification qu'une composition donnée est correcte vis-à-vis d'un besoin.

La première (et plus simple) vérification à effectuer est que les signatures des services sont compatibles, c'est-à-dire que les interfaces de sortie d'un service sont compatibles avec les interfaces d'entrée des services auxquelles elles sont liées.

Il peut également être intéressant de vérifier que la fonctionnalité réalisée par la composition est bien celle souhaitée. Bien sûr, la méthode de vérification dépendra étroitement de la description choisie. Deux approches peuvent être envisagées. Il faut d'abord définir un modèle de calcul des fonctionnalités d'une composition, puis calculer la fonctionnalité de la composition et enfin vérifier qu'elle est compatible avec la fonctionnalité souhaitée. Pour éviter cette mécanique compliquée, dans [AAM17] les auteurs proposent d'évaluer tous les chemins possibles et de vérifier s'ils sont compatibles avec la post-condition. C'est un problème non trivial similaire à la vérification de modèle.

La vérification de la composition ne se limite pas à la fonctionnalité rendue par la composition. La façon dont les services communiquent entre eux est également un aspect important d'une composition de services. Ce qui intéressera plus particulièrement est la capacité des services à dialoguer entre eux. Dans [BSBM04] les auteurs définissent des propriétés telles que l'absence d'interblocage ou l'absence de message éternellement sur le réseau. Cette problématique se complique quand les services interagissent de façon asynchrone. Dans le monde des web services, le problème classique est la vérification qu'une composition est conforme à une chorégraphie. La chorégraphie spécifie les ordres possibles dans lesquels les services peuvent échanger des messages. Il s'agit alors de s'assurer que la projection de la chorégraphie est bien compatible avec les services présents dans la composition et que ceux-ci interagissent de façon conforme à la chorégraphie [BZ09].

Des propriétés de qualité de service peuvent également être importantes à vérifier. Par exemple, dans les systèmes critiques, il peut être important de pouvoir assurer que le temps de calcul ne dépasse pas une certaine borne.

2.2.4 L'adaptation

Lorsque que la composition de service ne valide pas la propriété souhaitée / n'est pas conforme à la spécification attendue, il peut être nécessaire d'adapter la composition.

L'adaptation la plus simple est sans doute lorsque le type de sortie d'un service n'est pas compatible avec le type d'entrée du service avec lequel il est connecté. Un adaptateur est alors un convertisseur de type.

L'adaptation prend également tout son sens au niveau du comportement et beaucoup de travaux ont été réalisés dans ce domaine [EBMN12]. Deux approches peuvent être adoptées pour rendre les services compatibles : adapter les services eux-mêmes (par exemple en renommant des messages) ou adapter la composition (par exemple en générant un contrôleur qui va réordonner les messages [YS97]).

2.2.5 Positionnement

Les travaux présentés dans ce manuscrit abordent la découverte de compositions de services de calcul en s'intéressant à leur sémantique ; la sélection de la meilleure composition de services vis-à-vis de la qualité de service ; la validation des compositions vis-à-vis des interactions entre services et l'adaptation de la composition pour réordonner les interactions entre services.

La majorité des travaux présentés ici abordent la découverte de la composition en se basant sur la signature des services ainsi que leurs fonctionnalités. Néanmoins, la qualité de service apparaît dans certains travaux. Certains des travaux sur la découverte sont réalisés sans plan de la composition. Celui-ci est découvert à la volée, soit par une approche de chaînage (voir section 5.2.2), soit grâce à l'exploitation des spécificités de la description qui permettent de construire "naturellement" la composition (voir section 3.1). D'autres travaux sont réalisés avec un plan de composition déjà défini (voir section 5.2.3).

Les travaux sur la sélection portent uniquement sur un critère de qualité de service. Une première approche ne s'intéresse qu'à un seul critère : le temps d'exécution, et le modèle de coût de la composition est déduit grâce à des algorithmes d'apprentissage (voir section 5.1). Une seconde approche couple la découverte et la sélection, en utilisant plusieurs critères de qualité de service, agrégés ou non (voir section 5.2).

Les travaux de vérification portent uniquement sur le comportement (une partie des travaux sur la découverte ayant été prouvée correcte, il n'est alors pas nécessaire de refaire une phase de vérification de la composition). Ce qui est étudié est la capacité des services à communiquer entre eux dans le monde asynchrone (voir chapitre 4).

Enfin, l'adaptation est peu abordée dans ces travaux. Dans le cas où un ensemble de services n'est pas capable de communiquer, un ensemble de priorités sur les messages est inféré automatiquement pour que la composition garantisse une propriété souhaitée (voir section 4.5).

2.3 Résumé des travaux

Les travaux sont résumés brièvement dans la figure 2.1.

Ces travaux ont été réalisés en collaboration avec différents étudiants en thèse et master, ou différentes collaborations nationales ou internationales. Ces collaborations sont résumées dans la figure 2.2.

	Signature	Sémantique	Comportement	Qualité de service
Découverte	3.1 sans plan 3.3 avec plan 5.2 avec ou sans plan	3.1 sans plan TAA 3.3 avec plan méta-donnée + workflow		
Sélection				5.1algo. d'apprentissage temps d'execution 5.2 algo. d'optimisation multi critères
Vérification			4.3vérification de modèle système de transition 4.5vérification de modèle système de transition	
Adaptation			4.5 inférence système de transition	

FIGURE 2.1 – Positionnement des travaux

3.1		Mes travaux de thèse (et leurs suites) encadrés par Marc Pantel, sous la direction de Michel Daydé
3.2		
	3.2.1	Innovative Computing Laboratory (ICL) - The University of Tennessee - Knoxville - USA
	3.2.2	Institute for Informatics and Automation Problems - National Academy of Sciences of the Republic of Armenia - Yerevan - Arménie
	3.2.3	National Institute of Informatics (NII) - Tokyo - Japon
3.3		Projet Grid-TLSE
4.2		Thèse et master de Florent Chevrou (2014-2017)[1]
4.3		Thèse et master de Florent Chevrou (2014-2017)[1] Master de Nathanael Sensfelder (2016)[1] Master d'Adam Shimi (2017)[1]
4.4		Thèse et master de Florent Chevrou (2014-2017)[1]
4.5		Master de Nathanaël Sensfelder (2016)[1]
5.1		Information and Computer Sciences Department - University of Hawaii at Manoa - Honolulu - USA
5.2		Thèse de Seriel Boussalia (2014-2016)[2] MISC Laboratory - université de Constantine 2 - Constantine - Algérie

1. étudiant co-encadré avec Philippe Quéinnec.
2. étudiante co-encadrée avec Allaoua Chaoui.

FIGURE 2.2 – Collaborations

2.4 Organisation du manuscrit

Le manuscrit s'organise de la façon suivante.

Le chapitre 3 se concentre majoritairement sur les aspects de découverte de la composition de services dirigée par une fonctionnalité souhaitée. La section 3.1 définit un courtier de services de calculs qui découvre automatiquement une composition de services répondant aux besoins de l'utilisateur. Ce courtier s'appuie sur une description du domaine des services réalisée grâce à une signature hétérogène avec sous-typage (ensemble d'opérateurs typés, ensemble d'équations sur ces opérateurs et liens de sous-typage). La section 3.2 décrit l'intégration de ce courtier dans trois intergiciels de grilles. La section 3.3 présente le projet Grid-TLSE (site d'expertise en algèbre linéaire creuse) et les travaux relatifs à la description des solveurs (services) à l'aide de métadonnées et des scénarios d'exécution (composition de services) à l'aide de workflow.

Le chapitre 4 se concentre majoritairement sur les aspects de vérification de composition de services du point de vue des interactions entre les services. La particularité de ces travaux est de se placer dans le monde asynchrone en en explorant la diversité. La section 4.2 introduit les définitions d'exécutions et calculs distribués, ainsi que la notion de modèle de communication (média qui régit la communication). La section 4.3 définit un framework de vérification de compatibilité de service. Cette vérification (automatisée avec TLC le model checker de TLA$^+$) prend en paramètre le modèle de communication qui définit l'ordre de délivrance des messages ainsi que la propriété à vérifier. La section 4.4 définit formellement et de façon unifiée sept modèles de communication. Ces modèles sont ensuite raffinés avec des structures de données de différents niveaux d'abstraction. Ces modèles sont également comparés entre eux pour identifier lesquels permettent le plus d'indéterminisme dans l'ordre de délivrance des messages. Ces modèles sont génériques, c'est-à-dire que l'ordre de délivrance des messages dépend uniquement de leur ordre d'émission. Dans la section 4.5 des modèles applicatifs sont étudiés. Ils établissent des priorités de délivrance de certains messages par rapport à d'autres. Ces priorités peuvent être automatiquement inférées pour que la composition de services valide une propriétés LTL souhaitée.

Le chapitre 5 se concentre majoritairement sur les aspects de sélection de la composition de service dirigée par des critères de qualité de service. La section 5.1 utilise des algorithmes d'apprentissage pour choisir la meilleure composition issue du courtier, en terme de temps d'exécution. La section 5.2 combine des aspects de découverte et sélection en utilisant des algorithmes d'optimisation dans le but de guider la découverte de la composition vers la meilleure composition. La découverte de la composition est réalisée en se basant sur les signatures des services. Dans le cas de services simples (une entrée et une sortie), le plan de la composition est découvert par chaînage. Dans le cas de services complexes plus réalistes (plusieurs entrées et plusieurs sorties) le plan de la composition doit être donné.

Finalement le chapitre 6 discute des perspectives de ces travaux.

CHAPITRE 3

DÉCOUVERTE DES COMPOSITIONS DE SERVICES GUIDÉE PAR LA FONCTIONNALITÉ

Ce chapitre s'articule autour de la thématique de la découverte de la composition de services. Le but est de créer une composition qui réponde à un besoin fonctionnel.

Dans un premier temps un courtier de service de calculs est présenté. Son objectif est la découverte du service ou de la composition de service qui répond à un besoin. Il s'appuie sur une description du domaine des services à l'aide d'une signature hétérogène avec sous-typage et un ensemble d'équations caractérisant les opérateurs de celui-ci (proche type abstrait algébrique). Ce courtier est illustré en utilisant des bibliothèques d'algèbre linaire, les opérateurs décrivant le domaine étant alors l'addition, la multiplication, l'inversion ou encore la transposée de matrice(s).

Dans un second temps, l'intégration de ce courtier dans trois intergiciels de grille (le mot à la mode de l'époque !) est détaillée. Dans les deux premiers cas, il s'agit d'une intégration complète qui permet d'appeler le courtier à travers l'intergiciel. Dans le dernier cas, il s'agit de travaux plus prospectifs, qui s'intéressent à l'allocation des ressources physiques pour pouvoir déployer la composition de services générée par le courtier.

La dernière partie du chapitre aborde la description de la fonctionnalité des services sous un autre angle. Les travaux prennent place dans le projet Grid-TLSE qui a pour but de fournir un site d'expertise en algèbre linéaire creuse. Un des objectifs est de fournir à l'utilisateur un outil lui permettant de choisir le solveur d'algèbre linéaire le plus adapté à son problème et de l'aider à choisir ses paramètres. La particularité de ce travail est que tous les services (solveurs) réalisent la même fonctionnalité (résolution d'un système linéaire). Néanmoins, ils ne possèdent pas tous les mêmes paramètres, car les algorithmes ne sont pas identiques. Ces différences sont décrites à l'aide de méta-données. Le problème de l'utilisateur aussi appelé scénario (composition de services) est décrit à l'aide d'un workflow qui est composé de services et d'opérateurs de transformation de flux.

3.1 Les types abstraits algébriques pour le courtage sémantique

Dans certains domaines, comme le calcul scientifique ou le traitement d'un grand volume de données, les aspects fonctionnels et non fonctionnels sont très importants lorsqu'il s'agit de composer et d'exécuter des services. Le travail présenté ici se place dans le cadre de ces domaines. Le courtage fonctionnel est plus particulièrement étudié, dans le but de l'intégrer aux intergiciels de grille (comme Globus [FK97], XtremWeb[FGNC01], NetSolve [AAB+01], DIET [CDL+02], NINF [TNS+03], ITBL [FSYH03] ou encore CONDOR [TTL05]), ce qui permet de coupler un courtage fonctionnel précis et un courtage non fonctionnel performant. Le courtage fonctionnel associe à chaque service une description de sa sémantique, c'est-à-dire la fonctionnalité rendue, puis confronte la description du service recherché à l'ensemble des descriptions des services disponibles. Plusieurs cas de figures sont possibles. Un ou plusieurs services simples peuvent répondre partiellement ou totalement au problème. En cas de réponse partielle, la partie non couverte par ce service simple doit alors être réalisée par un service (ou une composition de services) et composée avec le service simple.

L'approche proposée repose sur une sémantique précise des services (dans le but de pouvoir exprimer le rôle des paramètres fonctionnels) disposant de procédures de comparaison semi-décidables. Ceci est possible en se plaçant dans des domaines d'application spécifiques pour disposer d'un nombre restreint d'opérations pour décrire les services et obtenir de meilleurs temps de réponse aux requêtes des utilisateurs. Les informations sémantiques sont spécifiées par des spécialistes du domaine applicatif qui ne sont pas des spécialistes du formalisme choisi pour exprimer la sémantique. Il faut donc que le traitement des informations soit totalement automatique.

De cette analyse rapide émergent les besoins suivants :

- La procédure de courtage doit être semi-décidable.
- La description d'un domaine d'application doit être réalisable par un spécialiste de ce domaine qui n'est pas spécialiste du formalisme choisi pour décrire les services.
- Le traitement de cette description doit être complètement automatisé.
- La description doit être assez précise pour que la comparaison puisse permettre l'extraction des valeurs des paramètres des services.

Dans un premier temps le mécanisme de recherche d'un service simple est présenté. Puis, le mécanisme de construction d'une composition de service est présenté.

3.1.1 Une description basée sur les types abstraits algébriques

Une description des services inspirée des techniques formelles de spécification et de développement du logiciel est proposée dans ces travaux. Un problème majeur se pose dans le cas

général : soit la sémantique est très pauvre, soit la comparaison n'est pas complète ou décidable (les algorithmes de comparaisons peuvent ne pas terminer).

Pour les raisons évoquées précédemment (précision, formalisme disponible, procédure de comparaison semi-décidable...), les spécifications algébriques et l'unification équationnelle sont exploitées pour résoudre le problème de la découverte d'une composition de services. Ces spécifications permettent de définir les fonctionnalités rendues par un service à l'aide des structures de données qu'il manipule. De plus, les notations algébriques sont proches des notations mathématiques, ce qui convient bien aux domaines d'application considérés : les bibliothèques de calcul scientifique, avec en premier lieu l'algèbre linéaire. Les descriptions par mots-clés posent un problème de signification qui est en partie levé ici car les notations standard des mathématiques sont utilisées pour représenter les termes. Une addition est notée "+", une soustraction "-",...

Les notations utilisées dans la suite sont celles de [LD90] et [GM92].

3.1.1.1 Formalisation du domaine applicatif

Un domaine est défini par une signature hétérogène avec sous-typage $(S,\leq,\Sigma)$ et un ensemble quelconque d'équations $\mathscr{E}$.

Une *signature* est un ensemble Σ muni d'une application $ar : \Sigma \to \mathbb{N}$. Si $f \in \Sigma$ et $ar(f) = n$, alors f est d'*arité* n, si $n = 0$ f est une *constante*.

Une *signature hétérogène* est une paire (S,Σ), avec S un ensemble de symboles de *sortes* et Σ un ensemble de symboles représentant des constantes et des fonctions typées :

- $c : s$, avec $s \in S$
- $f : s_1 \times ... \times s_n \to s_{n+1}$, avec $s_1 ..., s_n, s \in S$ et $n \geq 1$

On note $\Sigma_{w,s} = \{f | f : s_1 \times ... \times s_n \to s, \forall i \in [|1..n|], s_i \in S, w = s_1 ... s_n\}$.

Une *signature hétérogène avec sous-typage* est un triplet $(S,\leq,\Sigma)$ où (S,Σ) est une signature hétérogène, $(S,\leq)$ est un ensemble ordonné, et les opérateurs satisfont la condition suivante : $f \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2} \wedge w_1 \leq w_2 \Rightarrow s_1 \leq s_2$ où $w_1 \leq w_2 \triangleq w_1 = s_{1_1} ... s_{1_n} \wedge w_2 = s_{2_1} ... s_{2_n} \wedge \forall i \in [|1..n|] s_{1_i} \leq s_{2_i}$.

Une *équation* est composée de deux termes sur une signature.

Ici, S représente l'ensemble des types manipulés dans le domaine d'application (les matrices, les entiers... pour l'algèbre linéaire), $\leq$ est la relation de sous-typage sur les types. Σ est l'ensemble des opérateurs (addition, multiplication...) et des constantes (matrice nulle, matrice identité...) du domaine. $\mathscr{E}$ est un ensemble d'équations qui représentent les propriétés des opérateurs du domaine (associativité, commutativité, élément neutre...).

Tous les exemples sont développés sur un domaine simple défini comme suit : $S = \{Matrix\}$

$$
\begin{aligned}
&\Sigma = \{ \\
&\qquad 0, I \ : \rightarrow Matrix \\
&\qquad +, * : Matrix \times Matrix \rightarrow Matrix \\
&\} \\
&\mathscr{E} = \{ \\
&\qquad x : Matrix \quad x * I = x \quad , \quad x, y, z : Matrix \quad x + (y + z) = (x + y) + z \\
&\qquad x : Matrix \quad I * x = x \quad , \quad x, y, z : Matrix \quad x * (y * z) = (x * y) * z \\
&\qquad x : Matrix \quad x * 0 = 0 \quad , \quad x, y, z : Matrix \quad x * (y + z) = (x * y) + (x * z) \\
&\qquad x : Matrix \quad 0 * x = 0 \quad , \quad x, y, z : Matrix \quad (x + y) * z = (x * z) + (y * z) \\
&\qquad x, y : Matrix \quad x + y = y + x \ , \quad x : Matrix \quad x + 0 = x \\
&\}
\end{aligned}
$$

3.1.1.2 Formalisation des services

Les services disponibles sont regroupés dans des bibliothèques et l'ensemble des bibliothèques est consultable. Un service s possède un ensemble de paramètres noté P_s. La fonctionnalité rendue est exprimée par un terme de l'algèbre. Soit $\{p_1 ..., p_n\}$ l'ensemble des variables de P_s qui interviennent dans la description de la fonctionnalité. t_x est le type de x. Un service s est noté : $s(p_1 : t_{p_1}, ..., p_n : t_{p_n}) = r_s : t_{r_s}$, avec r_s un terme sur Σ, dont les variables appartiennent à P_s.

Exemple 3.1.1. *Exemples de services :*
$s_1(x : Matrix, y : Matrix, z : Matrix) = x + (y + z) : Matrix$
$s_2(x : Matrix, y : Matrix, z : Matrix) = x * (y + z) : Matrix$

Exemple 3.1.2. *Soit le service s suivant :*
$s(A : Matrix,\ B : Matrix,\ m : int,\ n : int,\ p : int) = A * B : Matrix$. *Il réalise le produit de la matrice A de taille $m \times n$ avec la matrice B de taille $n \times p$.*
Les paramètres m, n et p n'apparaissant pas dans la description du service, ils ne peuvent pas être déterminés par l'analyse.

Puisque la réponse du courtier doit permettre d'exécuter les services, la description a été étendue pour prendre en compte les paramètres non fonctionnels (voir 3.1.7.1). Mais aucune vérification statique n'est faite. Si l'utilisateur donne des données incohérentes, les erreurs seront détectées à l'exécution des services.

3.1.1.3 Formalisation de la requête

L'utilisateur souhaite résoudre un problème particulier à partir de ses propres données. C_r est l'ensemble des identificateurs qui permettent de désigner ses données. Il va alors exprimer son problème sous la forme d'une requête r qui est un terme sur Σ. Dans la suite, C_r est aussi nommé *constantes de la requête*.

Exemple 3.1.3. *Exemples de requêtes :*

$C_{r_1} = \{a, b : Matrix \quad\}, \; r_1 = a + b$

$C_{r_2} = \{a, b, c, d : Matrix\}, \; r_4 = a * b + c * d$

$C_{r_3} = \{a, b : Matrix \quad\}, \; r_3 = a * (b + I)$

3.1.2 Manipulation des théories équationnelles

3.1.2.1 Le problème à résoudre

Rappelons que dans un premier temps nous nous intéressons à savoir si un service peut résoudre le problème de l'utilisateur. Dans un second temps, si le service répond partiellement au problème, il sera composé avec d'autres services.

Pour connaître dans quelle condition le service s répond à la requête r, il faut être capable de calculer une substitution σ, telle que $r =_E \sigma(s)$. σ renseigne les valeurs à donner aux paramètres de s en fonction des constantes de la requête.

3.1.2.2 Comparaison grâce à la réécriture

Les techniques de réécriture sont une solution possible pour la résolution du problème posé.

La réécriture Le problème posé est la comparaison de deux termes modulo une théorie équationnelle $\mathscr{E}$. Une solution simple et élégante consiste à transformer $\mathscr{E}$ en un système de réécriture équivalent, fini et convergent (confluent et terminant). Le problème se ramènerait alors à l'égalité des deux formes normales. Pour prouver la propriété de convergence, il faut connaître des informations sur la théorie équationnelle, ce qui n'est pas le cas, puisqu'une des hypothèses de travail est l'absence de contraintes sur les équations. Néanmoins, il existe des méthodes pour transformer un système $\mathscr{E}$ fini mais quelconque en système de réécriture fini et convergent. La complétion de Knuth-Bendix [KB70] est un algorithme qui transforme un ensemble fini d'égalités en un système de réécriture fini, terminant et confluent, dont la réduction préserve l'égalité. Les entrées de cet algorithme sont la théorie équationnelle et une relation d'ordre bien fondée qui permet d'orienter les règles. Notons que cet algorithme est semi-décidable. Pour plus de détails sur les fondements de la réécriture voir [BN98].

Outils de réécriture considérés Les trois principaux outils de réécriture étudiés (en 2006) sont Elan [BKK+98], CîME [CMU04] et Maude [CDE+00, CDE+03] car ils proposent des fonctionnalités équivalentes, notamment une complétion de Knuth-Bendix et l'obtention d'une forme normale à partir d'un système fini et convergent. Néanmoins, CîME ne gérant pas les types et Elan n'acceptant pas le sous-typage, Maude a plus particulièrement retenu l'attention.

Maude permet de définir une signature hétérogène avec sous-typage. Maude propose une complétion de Knuth-Bendix qui permet donc d'obtenir un système fini et confluent, si la complétion de Knuth-Bendix termine avec succès. À partir d'un tel système, Maude permet, pour

chaque terme, d'obtenir une forme normale unique. La comparaison de deux termes se ramène donc à la comparaison de leur forme normale.

Maude possède également une commande *xmatch* qui permet de faire du filtrage modulo les propriétés d'associativité, de commutativité et d'éléments neutres. Malheureusement, elle ne permet pas de le faire pour des propriétés définies par des équations quelconques.

Les limites de cette approche Un prototype a été réalisé en utilisant Maude. Malheureusement, la nécessité de fournir un ordre bien fondé pour utiliser la complétion de Knuth-Bendix impose à l'utilisateur un minimum de connaissances en théorie de la réécriture. Cet ordre n'est pas toujours facile à trouver même pour un spécialiste du domaine. De nombreuses approches ont été étudiées pour engendrer automatiquement ou semi-automatiquement cet ordre mais aucune ne permet de couvrir tous les cas. Citons par exemple le *Recursive path ordering* introduit dans [Der82].

3.1.2.3 Unification équationnelle

Une autre approche pour rechercher des correspondances entre les services fournis et la requête (comparaison de deux termes), en tenant compte d'un certain nombre d'équations, est l'unification équationnelle et le filtrage équationnel. Les problèmes de filtrage et d'unification, simple ou modulo une théorie équationnelle, avec ou sans type, appartiennent à une catégorie de problèmes qui ont été beaucoup étudiés, car ils ont une place importante en informatique et notamment dans les assistants de preuve. Une bonne introduction à ces problèmes peut être trouvée dans [BS01].

L'unification Soit u et v, deux termes. Le problème d'*unification* est de savoir s'il existe une substitution σ, telle que $\sigma(u) = \sigma(v)$. Un tel σ est appelé *unificateur* de u et v. Le problème d'unification, du premier ordre, de deux termes est décidable. De nombreux algorithmes ont été proposés pour résoudre le problème d'unification dont certains en temps linéaire [MM82, PW76].

Le filtrage Soit u et v deux termes, le problème de filtrage est de trouver σ tel que $\sigma(u) = v$. σ est appelé un *filtre* de u et v. Si v ne contient pas de variable, les problèmes d'unification et de filtrage sont strictement identiques.

L'unification équationnelle Soit $\mathscr{E}$ une théorie équationnelle, u et v deux termes, σ est un *E-unificateur* de u et v si et seulement si $\sigma(u) =_{\mathscr{E}} \sigma(v)$. La propriété de décidabilité de l'unification n'est pas conservée dans le cas d'une théorie équationnelle quelconque. L'ensemble minimal complet d'E-unificateurs peut ne pas exister et s'il existe, il peut être infini.

Du fait de la perte de la décidabilité dans le cas général, de nombreux travaux ont été réalisés dans des théories particulières : associative [Mak77, Jaf90, Sch92], commutative [Sie79],

distributive [AT85, SS96], associative et commutative [Dom91]... Des travaux ont également été réalisés pour pouvoir combiner les algorithmes de différentes théories [BS96]. La principale restriction de ces algorithmes est qu'ils nécessitent des théories sur des symboles disjoints. Cela limite donc les cas où ces techniques peuvent être utilisées.

Le filtrage équationnel Le problème de *filtrage équationnel* de deux termes u et v a une solution si et seulement si il existe σ tel que $\sigma(u) =_{\mathcal{E}} v$. Comme pour l'unification équationnelle, dans le cadre d'une théorie équationnelle quelconque, il n'y a pas forcément d'existence d'un ensemble minimal de filtres [FH86]. Là encore des travaux ont été réalisés dans le cas de théories particulières. Par exemple [Con04] pour une théorie associative et commutative, [MD96] pour des systèmes de réécriture ou [DMS92] pour des systèmes convergents.

3.1.3 Savoir si un service répond à une requête

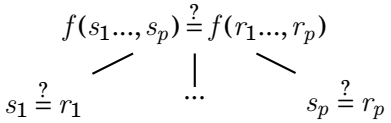
Rappelons que dans un premier temps, l'objectif est de savoir si un service répond à une requête. Le cas de la composition sera traité dans un second temps. Deux algorithmes sont proposés. Ils sont basés sur les travaux de Gallier et Snyder sur l'unification équationnelle [GS89] .

Le principe des deux algorithmes est de décomposer le problème initial en sous-problèmes, soit en appliquant des équations, soit en comparant les fils deux à deux. Chaque sous-problème engendre des contraintes sur la forme de la solution.

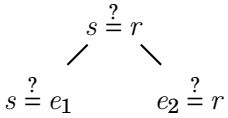
3.1.3.1 Un algorithme facilement parallélisable

Dans le premier algorithme, toutes les contraintes résultant du traitement des sous-problèmes doivent être obtenues avant d'être recombinées pour obtenir le résultat. Cela rend l'algorithme facilement parallélisable, puisque chaque sous-problème est traité indépendamment. Les deux règles principales se représentent sous la forme suivante :

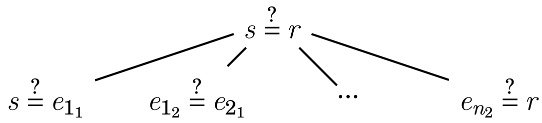
- Décomposition de termes



- Application de l'équation $e_1 = e_2$



ou application de la transformation (suite d'équations) $e_{1_1} = e_{1_2}; ...; e_{n_1} = e_{n_2}$:



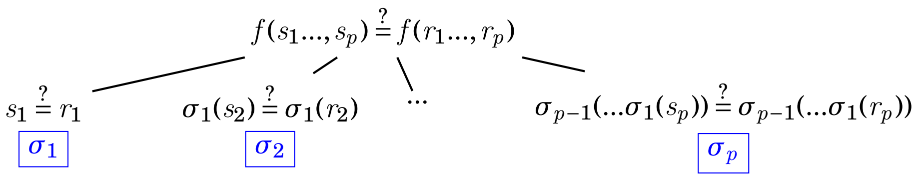
Il faut ensuite une étape de traitement des différents résultats des sous-problèmes pour s'assurer de leur cohérence (pas de variable associée à deux constantes différentes) et construire la solution (construire la substitution qui associe aux variables du service, les constantes de la requête).

3.1.3.2 Un algorithme plus efficace

Le second algorithme propage les contraintes au fur et à mesure de leur obtention et modifie ainsi les autres sous-problèmes. La facilité de parallélisation est perdue mais l'algorithme obtenu est plus efficace (détection des incohérences plus rapides).

Les deux règles principales se représentent sous la forme suivante :

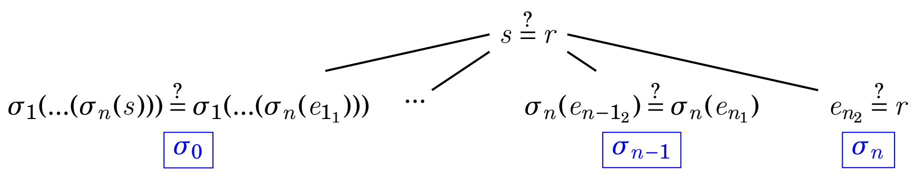
- Décomposition de termes



⇒⇒

Substitution associée : $\sigma_1 \circ ... \circ \sigma_p$

- Application d'équations $e_{1_1} = e_{1_2}; ...; e_{n_1} = e_{n_2}$



⇐⇐

Substitution associée : $\sigma_n \circ ... \circ \sigma_0$

3.1.3.3 Utilisation du système de types

Soit $s \stackrel{?}{=} r$ le problème à résoudre. Notons t_x, le type du terme x. La relation d'ordre utilisée sur les types est celle définie dans la signature hétérogène avec sous-typage, qui définit le domaine d'application.

Si $t_r \not\leq t_s$, le service est rejeté car il ne peut pas répondre à la requête.

Les transformations $e_{1_1} = e_{1_2}; ...; e_{n_1} = e_{n_2}$ ne sont pas construites aléatoirement, mais telles que :

- s et e_{1_1} ont le même symbole à la racine et $t_{e_{1_1}} \leq t_s$
- e_{x_2} et e_{x+1_1} ont le même symbole à la racine et $t_{e_{x+1_1}} \leq t_{e_{x_2}}$
- e_{n_2} et r ont le même symbole à la racine et $t_r \leq t_{e_{n_2}}$

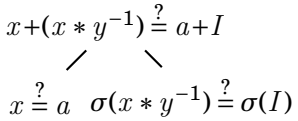
Cela permet de limiter le nombre de transformations applicables et donc d'accélérer la recherche de solutions.

3.1.3.4 Résultats

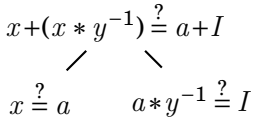
Obtention d'une solution Voici un exemple de résolution avec le second algorithme. En se plaçant dans le cadre de l'algèbre linéaire (avec ses opérateurs et propriétés usuels), le problème est de savoir si le service $s(x : Matrix, y : Matrix) = x + (x * y^{-1}) : Matrix$ peut répondre au problème suivant : $Matrix\ a : a + I$.

Dans la suite, σ est la substitution résultant du traitement des frères du nœud où elle apparaît, et qui doit être appliquée à un terme. Pour simplifier la lecture, elle sera toujours nommée σ même si elle est différente à chaque étape.

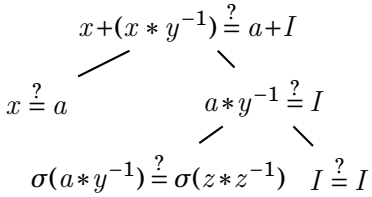
1. Décomposition (+ à la racine)



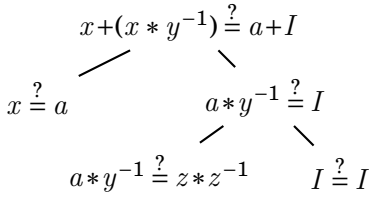
2. Application de $\{x \rightarrow a\}$



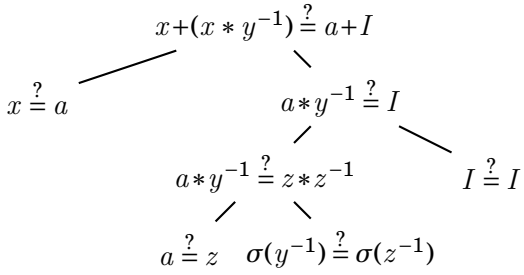
3. Application de l'équation $x : Matrix : x * x^{-1} = I$



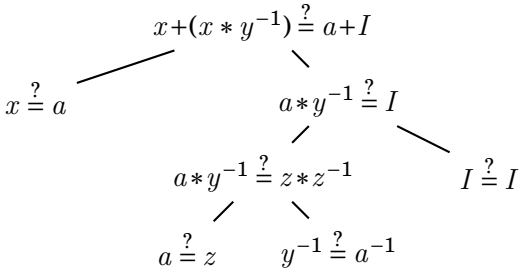
4. Application de la substitution (identité car $I = I$ n'impose pas de contrainte)



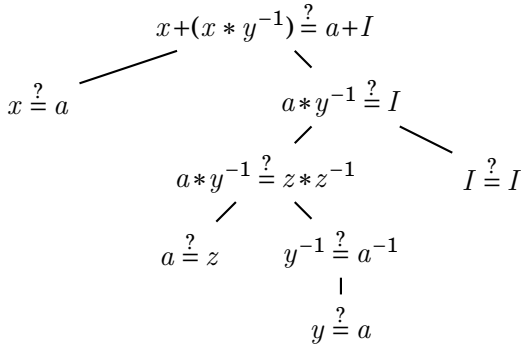
5. Décomposition



6. Application de $\{z \rightarrow a\}$



7. Décomposition



La substitution calculée est $\{x \mapsto a, y \mapsto a, z \mapsto a\}$ et donc la solution associée est $s(a,a)$.

Obtention d'un ensemble de solutions Plaçons-nous dans le cas du domaine précédent avec comme service disponible : $s_1(x : Matrix, y : Matrix, z : Matrix) = x + (y + z) : Matrix$.

Si la requête est $Matrix\ a, b\ : r_1 = a + b$, l'ensemble des solutions trouvées est :

- $(s_1, \sigma = \{x \rightarrow a, y \rightarrow b, z \rightarrow 0\})$ soit $s_1(a, b, 0)$
- $(s_1, \sigma = \{x \rightarrow b, y \rightarrow a, z \rightarrow 0\})$ soit $s_1(b, a, 0)$
- $(s_1, \sigma = \{x \rightarrow a, y \rightarrow 0, z \rightarrow b\})$ soit $s_1(a, 0, b)$
- $(s_1, \sigma = \{x \rightarrow b, y \rightarrow 0, z \rightarrow a\})$ soit $s_1(b, 0, a)$
- $(s_1, \sigma = \{x \rightarrow 0, y \rightarrow b, z \rightarrow a\})$ soit $s_1(0, b, a)$
- $(s_1, \sigma = \{x \rightarrow 0, y \rightarrow a, z \rightarrow b\})$ soit $s_1(0, a, b)$

3.1.4 La composition "gratuite"

Ajoutons $s_2(x : Matrix, y : Matrix) = x * y : Matrix$, regardons maintenant ce qui se passe si la requête est $Matrix\ a, b, c\ : r_2 = a + b * c$. L'algorithme trouve des solutions de la forme :

- $(s_1, \sigma = \{x \rightarrow a, y \rightarrow b * c, z \rightarrow I\})$ soit $s_1(a, b * c, 0)$
- $(s_1, \sigma = \{x \rightarrow b * c, y \rightarrow a, z \rightarrow I\})$ soit $s_1(b * c, a, 0)$
- ...

Ces solutions sont rejetées dans le cas où la composition n'est pas autorisée. Par contre pour obtenir une solution qui compose des services, il suffit de relancer l'algorithme précédent sur les termes complexes (qui ne sont pas des constantes). Dans le cas de cet exemple, l'algorithme est relancé avec comme requête $b * c$, ce qui donne comme solution $(s_2, \sigma = \{x \rightarrow b, y \rightarrow c\})$. Il ne reste plus qu'à combiner ces deux résultats pour obtenir la composition de services qui résout le problème soit $s_1(a, s_2(b, c), 0)$ par exemple. Dans le cas de solutions multiples pour le problème initial, toutes les recombinaisons sont effectuées, ce qui peut conduire à un grand nombre de

compositions envisageables. La composition est donc bien faite de façon dynamique, dans le sens où elle n'est pas connue à l'avance et que la construction de cette composition est guidée par la comparaison avec les services existants. Selon les services disponibles, la composition proposée ne sera pas la même, et plusieurs compositions peuvent être proposées. La méthode utilisée est à rapprocher de celle de [MA04], car l'algorithme est relancé sur le sous-problème qui correspond à la partie manquante. La composition est dite "gratuite", car aucun traitement spécifique n'est nécessaire pour l'obtenir. C'est le résultat de la comparaison d'un service et d'une requête qui donne naturellement la composition à effectuer.

3.1.5 Correction et complétude

Les algorithmes ont été prouvés corrects. L'algorithme facilement parallélisable a été prouvé correct en montrant qu'il appliquait une suite de règles d'inférence du système de Gallier et Snyder [GS89]. Ce système étant correct, l'algorithme l'est aussi. L'algorithme optimisé a été prouvé correct par récurrence sur l'arbre de transformations qu'il engendre.

L'algorithme facilement parallélisable n'est pas complet. Il applique les règles de transformations de Gallier et Snyder, mais impose une heuristique d'ordre d'application de ces règles, qui induit une perte de complétude. Néanmoins sans cette heuristique, l'algorithme n'est pas utilisable car beaucoup trop lent. C'est pour cette raison qu'un second algorithme a été développé.

L'algorithme optimisé a été prouvé complet, par récurrence sur le nombre d'équations appliquées.

3.1.6 Complexité

Le point faible de l'approche développée est son efficacité. Le calcul de la complexité du second algorithme a été réalisé dans [Hur06]. Voilà un résumé.

- n_s désigne le nombre de services initiaux ($n_s \neq 0$) ;
- n_{sol} désigne le nombre maximal de solutions obtenues ;
- n_e désigne le nombre d'équations ;
- m désigne le nombre maximal d'équations appliquées ($n_e \neq 0$ si $m \neq 0$) ;
- d désigne la profondeur maximale de composition ;
- p désigne le nombre maximal de paramètres des services ($p \neq 0$) ;

La complexité de l'algorithme dans les différents cas est :

	$m = 0$	$m \neq 0$
$d = 0$	$\mathcal{O}(n_s)$	$\mathcal{O}(m * {n_e}^m * n_s)$
$d \neq 0$ *et* $p * n_{sol} = 1$	$\mathcal{O}(n_s * d)$	$\mathcal{O}(d * m * {n_e}^m * n_s)$
$d \neq 0$ *et* $p * n_{sol} > 1$	$\mathcal{O}(p^d * {n_{sol}}^d * n_s)$	$\mathcal{O}(m * {n_e}^m * p^d * {n_{sol}}^d * n_s)$

Il est important de noter que l'algorithme est :

- linéaire par rapport au nombre de services (si n_{sol} est non corrélé avec n_s) ;
- exponentiel par rapport à la profondeur de composition (choix de l'utilisateur) ;
- polynomial par rapport au nombre d'équations ;
- exponentiel par rapport au nombre d'équations qui peuvent être appliquées (choisi par l'utilisateur).

Dans les exemples sur l'algèbre linéaire (environ 100 équations) avec des services issus du BLAS [BDD+02, DDDH90] et d'une partie LAPACK [ABB+99] (deux bibliothèques d'algèbre linéaire très répandues) en simple et double précision (environ 40 services), une profondeur de composition et un nombre d'équations appliquées (par niveau de composition) égaux à 6 étaient suffisants pour des résultats pertinents.

La profondeur de composition et le nombre d'équations appliquées sont des valeurs critiques, mais ce sont les mécanismes que ces paramètres contrôlent qui font la richesse de l'approche. Pour gagner en efficacité sans perdre les solutions intéressantes, il n'est donc pas possible de simplement donner des valeurs faibles à ces deux paramètres. Il faut cependant noter que le temps de calcul du courtage ne dépend pas de la taille des matrices, contrairement au temps d'exécution des services : plus les services manipulent des données importantes, plus la part du courtage est négligeable.

3.1.7 Illustration

Les différentes utilisations possibles de l'outil de courtage vont maintenant être illustrées. Il peut être utilisé seul, plutôt comme "documentation" mécanisée ou il peut aussi être intégré dans une plate-forme plus complète afin que les appels soient complètement cachés à l'utilisateur. Trois de ces intégrations sont développées dans la suite.

Tous les exemples sont donnés dans le cadre de l'algèbre linéaire. Une description du domaine est détaillée dans [Hur06]. Deux bibliothèques très répandues sont utilisées : le BLAS et un sous-ensemble de LAPACK.

3.1.7.1 Description des paramètres non fonctionnels

Dans le but d'une automatisation de l'exécution de la composition de services renvoyée par le courtier, il est indispensable d'avoir une valeur pour tous les paramètres, y compris les paramètres non fonctionnels. Une description des paramètres non fonctionnels a donc été intégrée. Cette description se limite aux paramètres non fonctionnels rencontrés dans les bibliothèques d'algèbres linéaires, à savoir des paramètres relatifs aux tailles des matrices et à la façon dont celles-ci sont stockées en mémoire.

Par exemple :

- value x = e_1 `||` e_2 signifie que la valeur du paramètre x est égale à l'expression e_1 ou e_2, ces deux expressions devant s'évaluer en valeurs égales ;
- value x = (nr A) (resp. (nc A)) signifie que la valeur du paramètre x est égale au nombre de lignes (resp. colonnes) de la matrice A.

3.1.7.2 BLAS et LAPACK

Dans un premier temps, les descriptions des services des deux bibliothèques qui interviennent dans la suite sont données.

BLAS BLAS (*Basic Linear Algebra Subprograms*) regroupe des procédures qui réalisent des opérations sur les matrices et vecteurs de façon performante. Le niveau 1 du BLAS s'intéresse aux opérations scalaire / vecteur et vecteur / vecteur. Le niveau 2 implante les opérations vecteur / matrice et le niveau 3 les opérations matrice / matrice. BLAS, efficace et portable, est couramment utilisé dans d'autres bibliothèques, comme c'est le cas pour LAPACK, qui propose des opérations de plus haut niveau.

Par exemple la procédure dgemm effectue une multiplication généralisée de matrices de la forme $C \leftarrow \alpha A \times B + \beta C$ (où A et B peuvent éventuellement être transposées). Elle est décrite comme suit pour qu'elle puisse être exploitée :

```
dgemm(transa:Char, transb:Char, m:Int, n:Int, k:Int, α:Real,
       A:Matrix, lda:Int, B:Matrix, ldb:Int, β:Real,
       C:Matrix, ldc:Int)
  C ← ((α * (op(transa,A) * op(transb,B))) + (β * C))
  value m = (nr C ) || (nr (op transa A) ) ;
  value n = (nc C ) || (nc (op transb B ) ) ;
  value k = (nc (op transa A) ) || (nr (op transb B ) ) ;
  value lda = (nr C) || (nr (op transa A ) ) ;
  value ldb = (nc C ) || (nc (op transb B ) ) ;
  value ldc = (nc (op transa A) ) || (nr (op transb B ) ) ;
```

Les quatre premières lignes correspondent à la description de la signature et de la fonctionnalité, les suivantes (`value .. = ...`) correspondent à la description des paramètres non-fonctionnels.

LAPACK LAPACK (*Linear Algebra PACKage*) est une bibliothèque d'algèbre linéaire qui contient des procédures pour résoudre les systèmes linéaires, le calcul des valeurs propres et des valeurs singulières. Les factorisations de matrices (LU, Cholesky,...) sont aussi fournies. Les matrices denses et par bandes sont traitées, mais pas les matrices creuses. Dans tous les cas,

les mêmes fonctionnalités sont fournies pour des matrices réelles ou complexes et en simple ou double précision.

LAPACK possédant un nombre très important de procédures (plus d'une centaine de procédures dans le cas des réels en simple précision), seule une partie de LAPACK a été décrite.

Voici la description des services qui apparaissent dans les exemples :

- dgetrf : factorisation LU pour des matrices générales,
- dlaswp : application d'une permutation sur les lignes d'une matrice,
- dgesv : résolution d'un système linéaire pour des matrices générales.

3.1.7.3 Exemple d'utilisation du courtier

Le courtier, utilisé seul, est un outil de documentation plus que d'exécution. Un spécialiste du domaine décrit le domaine concerné et les principales bibliothèques associées, l'utilisateur interroge le courtier qui lui indique quelles procédures de quelles bibliothèques permettent de résoudre son problème et comment les composer. L'utilisateur doit ensuite programmer les appels des services concernés.

Si l'utilisateur souhaite résoudre un système linéaire avec une matrice générale A et un vecteur de second membre B (requête : $A^{-1} * B$), l'algorithme trouve, entre autre, la solution suivante :

```
p1=A;
p2=(ipiv);
dgetrf(nr(p1),nc(p1), p1,nr(p1), p2, (info) );
  //p2<- factorisation LU (A= P*L*U)
p3=B;
dlaswp(nc(p3), p3,nr(p3), (k1), (k2), p2, 1 );
  //p3<- permutation des lignes de B
p4= p3;
dtrsm('l','l','n','n',nr(p4),nc(p4),1.0,p,nr(p1),p4,nr(p4));
  //résout L*x=p3; p4<-x;
p5= p4;
dtrsm('l','u','n','n',nr(p5),nc(p5),1.0,p1,nr(p1),p5,nr(p5));
  //résout U*x=p4; p5<-x;
p5;
```

Cela revient à composer quatre services : dtrsm(..dtrsm(..dlaswp(..dgetrf(..)))).
Cet exemple illustre le fait que le résultat est extrêmement précis sur les valeurs que doivent prendre les paramètres et la façon dont doivent être composés les services. A noter qu'il s'agit exactement des appels réalisés par la procédure expert dgesv de LAPACK qui enchaîne toutes les étapes de la résolution du système linéaire. Si cette procédure est disponible, elle fera également partie des solutions proposées.

Dans les mêmes conditions, une réponse à la requête $(a * b) * (c * d)$ est :

```
p1=0;
dgemm('n','n',nr(p1),nc(p1),nc(a),1.0,a,nr(a),b,nr(b),1.0,p1,nr(p1));
      //p1<-a*b + p1 = a*b
p2=0;
dgemm('n','n',nr(p2),nc(p2),nc(c),1.0,c,nr(c),d,nr(d),1.0,p2,nr(p2));
      //p2<-c*d + p2 = c*d
p3=0;
dgemm('n','n',nr(p3),nc(p3),nc(p1),1.0,p1,nr(p1),p2,nr(p2),1.0,p3,nr(p3));
      //p3<- p1*p2+p3 = a*b*c*d
p3;
```

On compose alors trois services : dgemm(..,dgemm(..),..,dgemm(..),..).

3.2 Intégration de l'outil de courtage dans d'autres outils

3.2.1 Gridsolve

Travaux en collaboration avec l'Université du Tennessee et plus particulièrement avec Asim YarKhan. Ils ont donné lieu aux publications suivantes : [LYD+13, HY10].

Dans le cadre d'une collaboration avec l'ICL (The Innovative Computing Laboratory at the University of Tennessee), le courtier a été intégré dans GridSolve [HY10, LYD+13].

3.2.1.1 Un aperçu de Gridsolve

Le but de Gridsolve est de créer l'intergiciel nécessaire pour fournir une passerelle entre les scientifiques ayant des calculs à effectuer et la grande quantité de services fournis par les architectures de grilles. Le but est que les utilisateurs, non expert des grilles, puissent bénéficier des facilités (en termes de processus partagés, stockage, logiciel...) qu'offrent les grilles de calcul. Gridsolve a été conçu pour permettre à une grande communauté de scientifiques, d'ingénieurs, de chercheurs et d'étudiants d'utiliser les grilles. En travaillant depuis l'ensemble des outils fournis par leur environnement de travail habituel, ils peuvent accéder aux ressources de la grille.

Gridsolve est un système client-agent-serveur (ou broker RPC) qui permet un accès distant à des ressources hardware et software, via une variété d'interfaces clientes.

Le système est composé de trois entités comme illustré sur la Figure 3.1.

- *Le client* a besoin d'exécuter des appels de procédures à distance. En plus des programmes C et Fortran, le client Gridsolve peut être un environnement interactif tel que Matlab ou Octave.

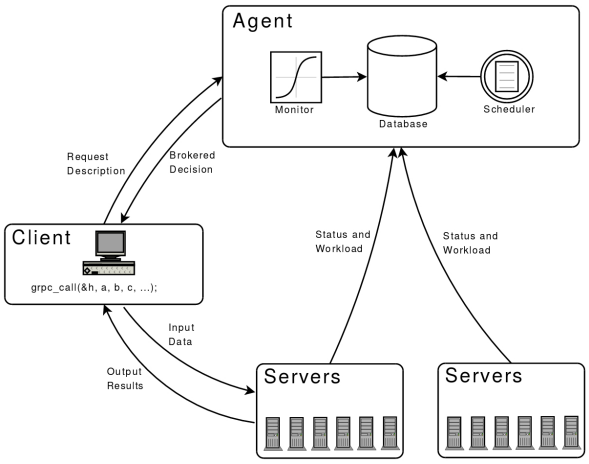


FIGURE 3.1 – Architecture de Gridsolve

- *Le serveur* exécute des fonctions selon les ordres des clients. L'architecture matérielle du serveur peut varier en complexité depuis le mono-processeur jusqu'à un serveur multiprocesseurs et les fonctions exécutées par le serveur peuvent être arbitrairement complexes.
- *L'agent* est le point clé du système Gridsolve. Il maintient une liste de tous les serveurs disponibles et réalise la sélection de ressources pour les requêtes du client. Il assure également la répartition de la charge de travail des serveurs.

3.2.1.2 Intégration du courtier dans Gridsolve

Le but est de faciliter le travail de l'utilisateur en rendant transparents les appels à la grille. Ainsi l'utilisateur n'a besoin d'aucune connaissance des ressources et services disponibles sur la grille :

1. Gridsolve fournit des informations à propos des services disponibles.
2. Le courtier trouve les combinaisons de services qui répondent à la requête de l'utilisateur.
3. La sortie du courtier est analysée et les services sont appelés.
4. La réponse est transférée à l'utilisateur.

Génération des entrées du courtier Gridsolve fournit, au courtier, les informations sur les services disponibles. Celui-ci a besoin d'informations sémantiques supplémentaires qui doivent être ajoutées à la description standard de Gridsolve. Ces informations supplémentaires sont les expressions mathématiques de la fonction calculée par le service ainsi que des informations sémantiques pour décrire le rôle des paramètres non-fonctionnels. Par exemple pour les fonctions *dgemm* et *dsymm* du BLAS, cela donnera :

```
SUBROUTINE dgemm
APPLICATION_DOMAIN="LinearAlgebra"
TRADER_DESCRIPTION="
c <-((alpha*((op transa a)*(op transb b)))+(beta*c)) ;
value m = (m c) || (m (op transa a)) ;
..."

SUBROUTINE dsymm
APPLICATION_DOMAIN="LinearAlgebra"
PARAMETERS_PROPERTIES = "a symmetric"
TRADER_DESCRIPTION="
c <- if (side='l') then ((alpha*(a*b))+(beta*c))
     if (side='r') then ((alpha*(b*a))+(beta*c)) ;
value m = (m c) ;
...
if (a instanceof UpTriInvMatrix ) then ( uplo = 'u' );
..."
```

Découverte des compositions de services L'utilisateur saisit sa requête (la forme varie selon l'interface choisie), Gridsolve invoque le courtier, qui lui renvoie un fichier contenant une séquence d'appels de service qui satisfait la requête. Par exemple, pour une requête $a + b + c$:

```
def, res2, copy c   //res2 <- copie de c
def, res1, copy a   //res1 <- copie de a
call, saxpy, m(b)*n(b), 1.0, copy b, 1, res2, 1  //res2 <- (copie de b) + res2
call, saxpy, m(res2)*n(res2), 1.0, res2, 1, res1, 1 //res1 <- res2 + res1
```

Gridsolve peut ensuite transformer cette séquence en un workflow DAG [LDSY08] pour améliorer les performances en évaluant différentes parties en parallèle quand cela est possible.

Appel des services Pour réaliser l'appel des services, le fichier de sortie du courtier est analysé et les appels GridRPC sont effectués avec les paramètres indiqués. Pour que ces appels soient possibles, des informations supplémentaires sont nécessaires : type des données, taille des données, pointeur sur la donnée... Selon l'interface client, ces données sont découvertes de façon différentes :

- l'utilisateur fournit les données dans l'interface C ;
- les informations sont retrouvées grâce aux mécanismes d'interrogation sur les données dans Matlab.

L'API C du courtier Le courtier s'appelle depuis l'API C de Gridsolve grâce à deux fonctions.

```
int gs_call_service_trader(char *req,... );
int gs_call_service_trader_stack(char * req, grpc_arg_stack *argsStack);
```

Le premier paramètre est la chaîne de caractères qui correspond à la requête utilisateur. Les autres arguments sont, pour chaque donnée utilisateur, son nom, un pointeur sur sa valeur, sa taille (nombre de lignes et nombre de colonnes).

Voici un exemple d'appel :

```
float *a = malloc (sizeof(float)*ma*na);
float *b = malloc (sizeof(float)*mb*nb);
gs_call_service_trader("(a+(b+a))","a",a,ma,na,...);
```

L'interface Matlab Le courtier peut également être appelé dans Matlab grâce à l'interface Matlab de GridSolve. L'interface est un peu plus simple puisque les informations sur les données utilisateur n'ont pas à être fournies mais sont obtenues en interrogeant Matlab. Voici un exemple d'appel depuis l'interface Matlab :

```
a=[1,2,3;4,5,6;7,8,9]
b=[10,20,30;40,50,60;70,80,90]
[output]=gs_call_service_trader("(a+(b+a))"),
output =
   12. 24. 36.
   48. 60. 72.
   84. 96. 108.
```

La variable de retour est intégrée dans le workspace Matlab et peut bien sûr être utilisée dans des calculs futurs.

3.2.1.3 Expérimentations avec Grid5000

GridSolve, avec le courtier, a été déployé sur Grid5000 [1]. L'agent est lancé sur une machine, par exemple `paramount-13.rennes.grid5000.fr` (machine du cluster Dell `paramount`, carte Myrinet de 10GB), les services sont déployés sur des machines du même cluster. L'utilisateur souhaite ajouter une matrice C à la solution d'un système linéaire ($A * X = B$) où A est une matrice quelconque. L'utilisateur demande donc de résoudre $A^{-1} * B + C$ au courtier. L'appel GridSolve se fait de la façon suivante : **gs_call_service_trader("**$(((inv\ a) * b) + c)$**...)**.

- Dans un premier temps, le BLAS est déployé sur `paramount-21.rennes.grid5000.fr`, le courtier ne trouve pas de solution car $dtrsm$, qui est le seul service qui résout un système linéaire, ne sait traiter que les matrices triangulaires.

1. https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

- LAPACK est ensuite déployé sur `paramount-2.rennes.grid5000.fr`, le courtier trouve une solution, qui compose *dgesv* (LAPACK, résolution de systèmes linéaires pour des matrices quelconques) et *daxpy* (BLAS, addition de matrices). Cette solution est calculée en moins d'une demi-seconde. La solution est alors exécutée en réalisant la composition qui met en jeu des services se trouvant sur deux machines différentes. L'utilisateur n'a pas à interagir, il récupère son résultat, une fois les calculs terminés.
- Le BLAS est ensuite déployé sur `paramount-23.rennes.grid5000.fr`, la même solution que précédemment est alors trouvée, dans le même délai de temps. C'est GridSolve qui gère le fait que le service *daxpy* est disponible sur deux machines différentes et qui choisit la machine où exécuter le service en fonction de leur charge.
- `paramount-21.rennes.grid5000.fr` et `paramount-23.rennes.grid5000.fr` disparaissent, le courtier n'est plus capable de trouver une solution (LAPACK n'ayant pas été déployé intégralement, il n'y a pas de service qui permette de faire l'addition).
- `paramount-21.rennes.grid5000.fr` réapparaît, mais avec seulement le BLAS de niveau 3, c'est-à-dire sans *daxpy*, le courtier trouve une solution qui compose *dgemm* et *dgesv*. Cette solution est exécutée sans que l'utilisateur ne se rende compte qu'il s'agit d'une solution différente de la précédente. Cette solution était d'ailleurs possible précédemment, mais non choisie, car plus coûteuse.

Ceci illustre un avantage du courtier associé à un intergiciel de grille (ou plus généralement à un système de découverte de ressource) : composition dynamique en fonction des services disponibles. Il est possible de remplacer un service par un autre plus général (et souvent plus coûteux) ou par une composition de services si ce service est amené à disparaître.

3.2.2 gLite et P-Grade

Travaux en collaboration avec l'Institute for Informatics and Automation Problems d'Arménie et plus particulièrement avec Hrachya Astsatryan. Ils ont donné lieu aux publications suivantes : [ASS+13, ASS+12, ASS+08, ASS+11]

Dans le cadre d'une collaboration avec l'Institute for Informatics and Automation Problems d'Arménie, le courtier a été intégré dans un service de calcul scientifique déployé sur une grille de calcul grâce au portail P-Grade [2] (Parallel Grid Run-time and Application Development Environment) et au middleware gLite.

3.2.2.1 Un aperçu de P-Grade et gLite

P-Grade P-Grade est un environnement web mettant à disposition un certain nombre de services pour le développement, l'exécution et le monitoring d'applications basées workflow. P-

2. http://www.p-grade.hu

Grade cache les mécanismes d'accès de bas niveau à la grille grâce à des interfaces graphiques de haut niveau, rendant les non-experts des grilles capables de définir et d'exécuter des applications distribuées sur des infrastructures de calculs multi-institutions. Le portail P-Grade peut accéder à de multiple grilles simultanément, ce qui facilite la distribution d'applications complexes sur plusieurs plate-formes.

gLite gLite est (était ?) un middleware pour les grilles dont la première version est sortie en 2005. Il a été développé à partir de composants pré-existants tels que ceux de Globus. Il fournit un framework pour développer des applications qui s'exécutent de façon distribuée et avec un stockage des données dans tout Internet.

Il fournit cinq grandes familles de services pour l'accès aux grilles, pour la sécurité (authentification, autorisation...), pour le monitoring, pour la gestion des données et pour la gestion des jobs.

3.2.2.2 Intégration du courtier dans P-Grade

L'objectif est de fournir un environnement de calcul scientifique qui permet de faire une passerelle entre les algorithmes d'algèbre linéaire et les intergiciels. Cela permet à des non-experts un accès simple aux services de calcul scientifique et à leur exécution sur une grille de calcul pour de meilleures performances.

L'environnement se décompose en quatre couches principales :

- l'interface utilisateur : elle permet à l'utilisateur de manipuler ses données (initialisation, upload, suppression...) et d'entrer le calcul souhaité en utilisant une syntaxe à la Matlab (qui est proche du langage d'entrée du courtier).
- un analyseur et le courtier : l'analyseur (code PHP) réalise une analyse syntaxique et sémantique de la requête de l'utilisateur et la transforme en un format lisible par le courtier. Celui-ci transforme la requête en une liste d'appels de procédures.
- un module de préparation des jobs : il convertit la sortie du courtier en une liste d'appels locaux ou distants (en fonction de la localisation des services et de la taille des données quand les services sont disponibles à plusieurs endroits). Lorsque la cible d'exécution est une infrastructure de grille, un fichier JDL (Job Definition Language) est généré pour spécifier les ressources nécessaires à un job. Ce fichier est ensuite soumis à la grille via le portail P-Grade, ce qui déclenche le lancement des calculs. C'est également ce module, qui se charge de remonter les résultats à l'utilisateur.

En conséquence, toutes les difficultés inhérentes à l'utilisation du middleware gLite, à la programmation MPI, ou la construction d'appel à des librairies complexes sont totalement cachées à l'utilisateur.

3.2.2.3 Expérimentations

Des expérimentations ont été faites en déployant plusieurs librairies de calcul scientifique, pour de l'algèbre linaire dense ou creuse. Ces librairies couvrent un grand nombre d'architectures, comme les architecture parallèles en utilisant OpenMP ou MPI, ou la programmation GPU. Les services déployés dans l'environnement sont les nombreuses procédures de ces librairies scientifiques.

Les librairies considérées sont : BLAS, ScaLAPACK (librairie d'algèbre linéaire pour des machines à mémoire distribuée), MAGMA (librairie dont la spécificité est de traiter des architectures hétérogènes comme par exemple les système multi-cœurs + GPU), PSBLAS (version du BLAS pour des matrices creuses et avec une parallélisation du code) et MUMPS (solveurs pour matrices creuses).

3.2.3 NAREGI

Travaux en collaboration avec le National Institute of Informatics (NII) de Tokyo - Japon et plus particulièrement avec Kento Aida.

Contrairement aux travaux précédents (GridSolve et PGrade), il ne s'agit pas d'intégrer directement le courtier dans l'intergiciel de grille. Le but est de réaliser un matchmaker, c'est-à-dire un outil qui s'occupe de la sélection des ressources, correspondant à la demande de l'utilisateur, pour exécuter une requête sur la grille. Le matchmaker est développé pour l'intergiciel de grille NAREGI, et permet d'affecter des ressources aux différents appels de services des solutions issues du courtier.

3.2.3.1 Un aperçu du projet Naregi

Le projet NAREGI [3] (National Research Grid Initiative) a pour but de fournir un intergiciel de grille. Le projet est en collaboration avec le laboratoire de nanoscience et leurs applications sont la cible du projet.

L'architecture de l'intergiciel de NAREGI est présentée dans la figure 3.2. La partie verte représente l'environnement utilisateur. Il est composé de trois composants et permet à l'utilisateur de lancer simplement et efficacement ses applications sur la grille. Le Workflow Tool permet de contrôler les jobs et c'est dans ce module que le matchmaker s'intègre. Le Grid Problem Solving Environment répartit les applications développées par les chercheurs sur l'environnement de grille. Enfin, le Grid Visualization System permet de visualiser le résultat des calculs.

3. https://www.naregi.org/project/index_e.html

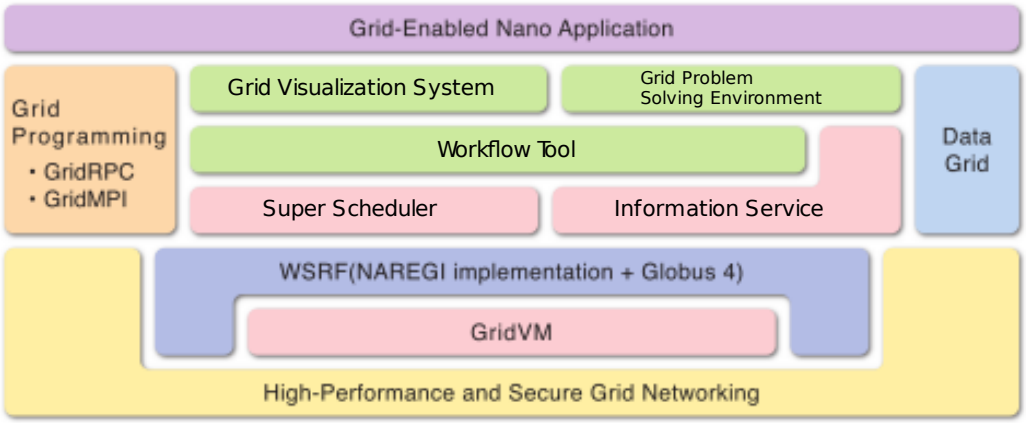


FIGURE 3.2 – Architecture de l'intergiciel NAREGI

3.2.3.2 Matchmaker basé ontologies

Dans le contexte des grilles de calcul, l'utilisateur peut faire tourner ses applications sur différents systèmes ; dans des lieux géographiques différents, avec des propriétaires et politiques différentes, sur des machines avec des caractéristiques (architecture CPU, système d'exploitation, mémoire, nombre de CPU...) variées. L'utilisateur peut avoir besoin d'une ressource présentant des caractéristiques bien précises pour son application. L'opération de trouver les ressources qui correspondent au besoin utilisateur est appelé *matchmaking*.

Comme dans les travaux précédents, la première nécessité est de décrire les ressources et les besoins utilisateurs. Traditionnellement les mécanismes de matchmaking sont basés sur des descriptions à base d'attributs comme pour le matchmaker Condor [RLS98]. Cela impose que le propriétaire de la ressource et l'utilisateur se mettent d'accord sur les attributs et valeurs. Cela rend les extensions des descriptions compliquées. Les ontologies permettent de définir des descriptions avec plus de sémantique et de façon plus flexible.

A l'aide de Protégé [4][KFNM04] et du standard OWL [5] une ontologie pour définir les ressources et les requêtes utilisateurs a été développée. Un algorithme de matchmaking a également été défini.

La requête utilisateur La requête utilisateur est composée de deux parties : le workflow potentiellement complexe (job en séquence, parallèle, boucle...) et l'utilisateur. Dans l'ontologie, nous retrouvons donc la partie workflow avec pour décrire les jobs unitaires toutes les informations qui sont trouvables dans le format JSDL [ABD+05] ainsi que des classes spécifiques pour décrire les workflows complexes. Les jobs unitaires sont caractérisés par une partie application (chemin de l'exécutable, date de début, durée...) et une partie ressources (machines candidates,

4. http://protege.stanford.edu/
5. http://www.w3.org/TR/owl-features/

système d'exploitation, architecture CPU, mémoire nécessaire...). L'ontologie permet également de résoudre le problème classique des unités d'expression des valeurs numériques.

Les ressources Dans le projet NAREGI, les ressources sont décrites sur le schéma CIM [6] [DMT99]. L'ontologie définie reprend donc ces informations avec quelques extensions : ajout des clusters, des familles d'OS et de processeurs, des logiciels présents sur les ressources. Une autre information ajoutée dans l'ontologie est la notion de politique d'utilisation pour un utilisateur. Par exemple, nous pouvons imaginer que tous les utilisateurs n'ont pas les mêmes droits en terme de temps d'utilisation, de nombre de CPU alloués ou en mémoire (pour raison de budget ou de partage). La notion de groupe d'utilisateurs a également été introduite et liée aux politiques d'utilisation.

L'algorithme de matchmaking L'algorithme de matchmaking compare les besoins de l'utilisateur avec toutes les ressources disponibles et trouve celles qu'il est autorisé à utiliser et qui répondent à ses besoins. Il commence par découper les workflows complexes pour s'intéresser aux jobs unitaires, puis recombine les résultats. Pour les jobs unitaires, plusieurs points sont vérifiés : la non violation par l'utilisateur des politiques d'utilisation, l'installation de l'application sur le système et la correspondance des caractéristiques souhaitées pour les ressources.

3.2.3.3 Bilan

Alors que le courtier s'intéresse à la découverte des compositions de services répondant à un besoin fonctionnel, le matchmaker s'intéresse à la découverte des machines sur lesquelles déployer la composition. Les machines ayant un ensemble bien définis de caractéristiques physiques, la description de celles-ci est relativement classique puisque issue du format JSDL. Certaines caractéristiques non présentes dans JSDL ayant été ajoutées (cluster, politique d'accès, ...), une description à l'aide d'une ontologie a été choisie pour permettre de formaliser ces notions, et ainsi décrire les besoins et ressources sans ambiguïté. Contrairement au courtier, les descriptions des ressources et besoins étant relativement simples, l'algorithme de recherche ne présente aucune difficulté.

3.3 Grid-TLSE - description basée sur un langage de workflow

Ces travaux ont eu lieu dans le cadre du projet Grid-TLSE. Ils ont donné lieu aux publications suivantes : [CDD+05, CCG+13]

Le projet Grid-TLSE a pour but de fournir un site d'expertise en algèbre linéaire creuse. Il a été financé successivement par :

6. http://www.dmtf.org/standards/cim/

- ACI "Globalisation des Ressources Infomatiques et des Données"
- Projet ReDIMSoPS via la coopération CNRS/JST (Japon)
- Projet SOLSTICE (ANR-06-CIS6-010)
- Projet LEGO (ANR-CICG05-11)
- Projet FP3C, projet collaboratif entre la France et le Japon (ANR-JST FP3C)
- Projet COOP (ANR-09-COSI-001)

Les partenaires initiaux étaient :

- les partenaires académiques : CERFACS, IRIT, LaBRI et LIP-ENS Lyon
- les partenaires industriels : CEA, CNES, EDF et IFP

Il a deux facettes. Un premier objectif est de permettre d'échanger des problèmes / matrices, en facilitant l'accès à des collections de matrices publiques, en permettant à l'utilisateur de télécharger ses matrices (tout en lui laissant le choix de les rendre publiques ou non) et en créant des groupes privés pour échanger des matrices. Le second objectif est de permettre à l'utilisateur de tester des solveurs sur des matrices (issues de cette base de données) sans qu'il ait besoin d'installer ces solveurs, ni même besoin d'acheter les licences de ces solveurs. Le but est d'aider les utilisateurs à choisir les bons solveurs et les bons paramètres pour ces solveurs afin de résoudre un problème donné. Les travaux présentés ici s'intéressent plus particulièrement à ce second objectif du projet.

3.3.1 L'architecture de la plateforme

L'architecture de la plateforme Grid-TLSE est présentée dans la figure 3.3.

- **WebSolve** est une interface web depuis laquelle l'utilisateur peut déposer ses matrices, soumettre ses problèmes et consulter les résultats de ses expérimentations.
- **Weaver** convertit la requête de l'utilisateur en plusieurs appels élémentaires aux solveurs. Pour ce faire, il s'appuie sur la description des scénarios (descriptions réalisées avec **Geos** et stockées dans une base de données) et sur les méta-données associées aux matrices et solveurs (description réalisée avec **Prune** et stockée dans une base de données).
- **Gridcom** est en charge du déploiement et de l'exploitation des solveurs sur la grille.
- DIET [DIE] est l'intergiciel de grille utilisé.

Les parties qui nous intéressent particulièrement dans ce manuscrit sont Weaver, Geos et Prune.

3.3.2 Description des scénarios à l'aide de workflows

Les scénarios (composition de services) sont définis avec un langage de workflow. Ce langage possède trois catégories d'opérateurs : les OpExec qui exécutent un solveur (service) sur tous les

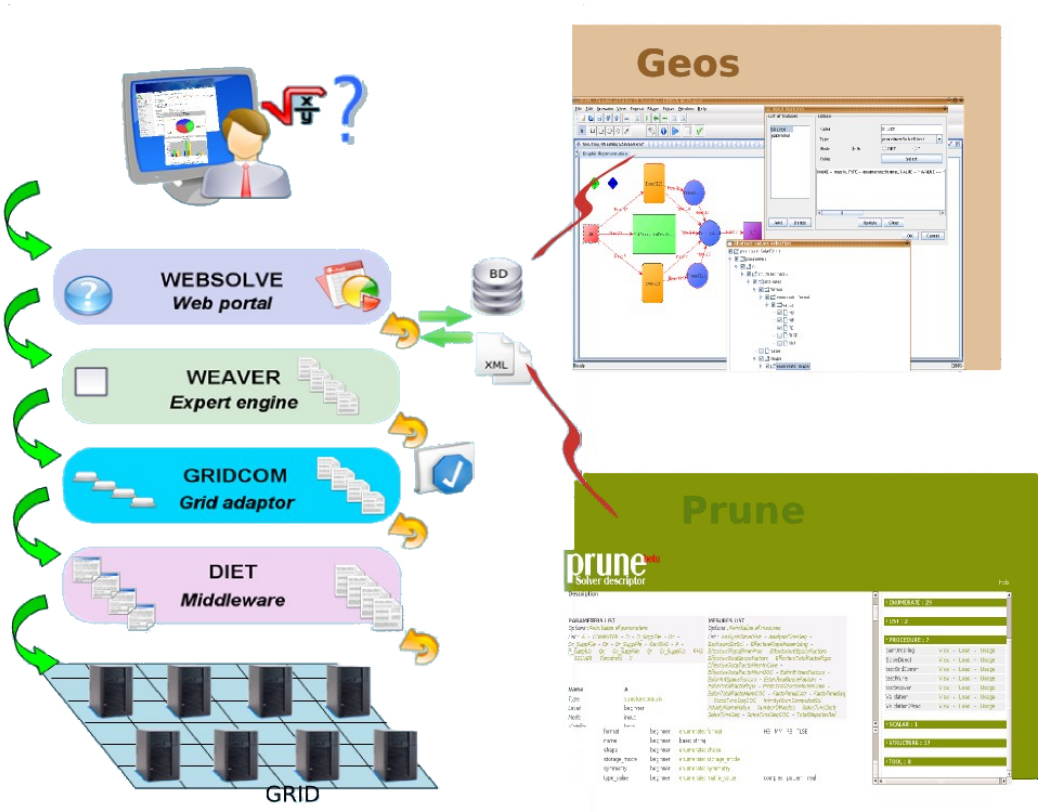


FIGURE 3.3 – Architecture de la plateforme Grid-TLSE

problèmes du flux entrant, les OpTrans qui transforment les flux, et les sous-scénarios. Sur les flux circule une liste d'expériences. Les OpTrans permettent de manipuler les flux d'expériences de diverses manières : union, intersection, fusion... de plusieurs flux, modification de toutes les expériences (positionnement d'un paramètre par exemple), filtrage en fonction des valeurs des paramètres...

Un exemple de scénario est donné dans le figure 3.4. Le but est de comparer le comportement d'un solveur sur une matrice A et sur une matrice B correspondant à la mise à l'échelle (scaling) de A (B = D_r A D_c). Il y aura donc trois branches au scénario : la matrice B (scaling étudié), la matrice A avec le scaling par défaut et la matrice A sans scaling. La comparaison des deux premières branches donne une indication sur la qualité de l'algorithme de scaling à tester. La dernière branche est le témoin. En entrée du scénario sont donnés la matrice A et le solveur. La branche du haut calcule dans un premier temps D_r et D_c. Comme en sortie du sous-scénario ne sont présents que D_r et D_c, la matrice A et le solveur sont réinjectés à l'aide d'un OpTrans, puis la matrice B est calculée. Le flux de sortie est alors ajouté à celui venant directement des entrées. Un OpTrans précise qu'il ne doit pas y avoir de scaling appliqué lors du calcul. Le tout est ensuite dirigé vers un OpExec qui réalise le calcul. Le flux de sortie de celui-ci est ajouté à

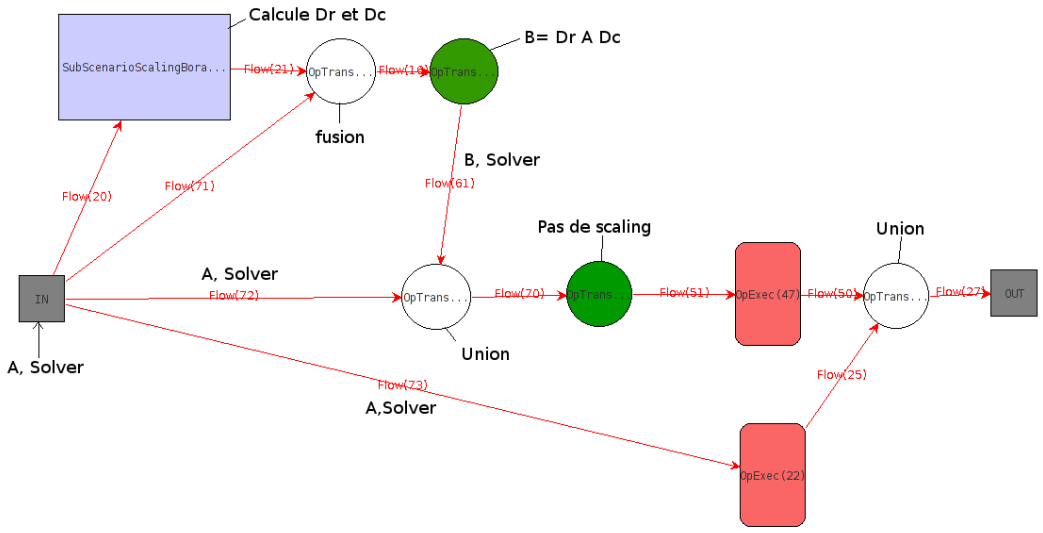


FIGURE 3.4 – Exemple de scénario de la plateforme Grid-TLSE

celui d'un autre OpExec où aucune transformation n'a été réalisée sur les données en entrée (et donc la matrice A sera traitée en utilisant le scaling par défaut).

Pour définir ces scénarios, des connaissances sur les solveurs et les matrices sont nécessaires. Par exemple pour pouvoir indiquer que l'on ne souhaite pas de scaling, il ne faut utiliser que des solveurs pour lesquels il est possible d'avoir cette option.

3.3.3 Description des matrices et des solveurs basée sur des méta-données

Tous les solveurs sont des solveurs directs (en opposition aux solveurs itératifs) capables de résoudre un système linaire creux. Ils sont donc tous similaires en terme de fonctionnalité rendue. Néanmoins ils utilisent des algorithmes différents et possèdent donc des paramètres de contrôle très variés. Comme l'objectif du projet est d'aider l'utilisateur à choisir le bon solveur et les bonnes valeurs de paramètres, il faut être capable de réaliser une comparaison des solveurs. Comme conséquence, tous les solveurs doivent présenter la même interface aux scénarios. Une solution simple serait d'écrire des enveloppes (wrappers) pour chaque solveur. Ces enveloppes prendraient les mêmes paramètres et renverraient les mêmes résultats. Cela peut être assez coûteux en terme de développement et très sensible à l'addition de nouveaux paramètres. Il faut aussi noter que l'interface commune des solveurs est assez pauvre.

Une approche plus intéressante repose sur l'utilisation de méta-données qui décrivent, pour chaque solveur, tous les paramètres et résultats possibles (ainsi que leurs valeurs possibles en cas d'énumération). Les solveurs sont décrits par la liste de leur paramètres (nom et type), la liste de paramètres de retour (en plus du résultat du calcul, les solveurs directs retournent un

ensemble de mesures : temps de calcul, flops. . .) et un certain nombre de contraintes (toutes les combinaisons de paramètres n'étant pas possibles, il faut décrire celles qui le sont). Les types sont également décrits dans Prune, à l'aide d'attributs, qui peuvent eux-même être des types complexes. Pour les types également, des contraintes apparaissent. Par exemple une matrice dont l'attribut "symétrie" est positionné à "hermitien" doit forcément avoir l'attribut "type des valeurs" positionné à "complexe".

3.3.4 Contributions au projet

J'ai intégré le projet alors qu'il avait plusieurs années d'existence, la plupart des choix techniques étaient déjà faits. J'ai donc principalement ajouté de nouvelles fonctionnalités en agissant sur les descriptions des solveurs et matrices, et en enrichissant le langage de description de scénario.

Dans un premier temps, je suis intervenue au niveau des contraintes. Il faut pouvoir les décrire facilement (les contraintes étant nombreuses, il ne faut pas que ça soit fastidieux à écrire) mais également pouvoir les exploiter facilement. En effet, elles doivent être prises en compte lors de la génération des exécutions sur la grille. Par exemple, sur le scénario précédent, si le solveur n'est pas donné explicitement mais que l'on veut tester tous les solveurs, il ne faut sélectionner que ceux qui ont la possibilité de choisir le scaling. Dans d'autres cas, il faut regarder les propriétés de la matrice pour pouvoir positionner certains paramètres du solveur. Le choix a été fait de représenter les contraintes sous forme arborescente. Un nœud contient la valeur d'un paramètre (ou d'un attribut pour les types), les branches donnent les ensembles de combinaisons de paramètres (attributs) possibles. Cette représentation a l'avantage d'être plus concise à écrire que l'ensemble de contraintes linéarisées. Par contre, pour un ensemble de contraintes, il y a plusieurs représentations possibles. Au concepteur de choisir la plus adaptée.

3.4 Bilan

Dans ce chapitre deux approches ont été présentées : un courtier de services de calcul et un site d'expertise en algèbre linéaire creuse.

Ces travaux sont les plus anciens de ce manuscrit. Ils ont débuté il y a une douzaine d'années. Les outils et techniques utilisés à l'époque ne seraient probablement pas ceux choisis aujourd'hui pour les réaliser. Par exemple, les méthodes et outils propres aux ontologies sont aujourd'hui matures alors qu'ils étaient balbutiants il y a douze ans.

Les travaux présentés s'articulent autour d'un domaine d'application bien précis à savoir l'algèbre linéaire (creuse). Ce domaine particulier à l'avantage d'être restreint et étudié depuis des années. Mais surtout il offre la possibilité de définir mathématiquement (et donc formellement) les opérations réalisées par les différents services. Ces descriptions sont sans ambiguïté car les notations mathématiques ont été unifiées. En effet si A et B sont des matrices, il n'y a

aucun doute sur ce que $A \times B$ ou encore A^T signifie. Les travaux sur le courtier se basent sur cette particularité. Il est néanmoins envisageable de les appliquer à d'autres domaines, proches des mathématiques, possédant également des opérateurs et propriétés bien connus, comme par exemple l'optimisation ou le traitement d'image. En effet, même si l'approche est générique, il faut pouvoir définir une signature hétérogène avec sous-typage pour le domaine applicatif des services, ce qui est bien adapté aux mathématiques, mais n'est pas toujours naturellement possible.

Une particularité du courtier est la description précise du rôle de chaque paramètre du service, qui permet à la fois de s'abstraire des problèmes d'ordre des paramètres dans les signatures, mais surtout de trouver la valeur de chaque paramètre. Cette description du rôle de chaque paramètre permet d'avoir la composition de service "gratuitement". En effet, si la valeur affectée à un paramètre n'est pas une constante de l'utilisateur, mais un terme complexe, cela signifie qu'il est nécessaire d'utiliser un autre service (ou composition de services) pour calculer la valeur souhaitée. Ainsi le plan de la composition est découvert à la volée. Cela permet une automatisation complète du processus de recherche, jusqu'à l'exécution de la composition de services, sans intervention humaine, comme le prouve l'intégration dans les différents intergiciels de grille.

Le point faible du courtier est sa complexité. Même si le développement n'a jamais été optimisé, car il a été développé dans une optique de preuve de concept, la complexité intrinsèque du problème d'unification équationnelle fait que le courtier ne passe pas à l'échelle avec un domaine complexe (grand nombre d'opérateurs et d'équations) et un grand nombre de services. Pour répondre à cette problématique, une version parallélisable de l'algorithme a été proposée, mais n'a jamais été implantée en tirant profit de cette possibilité. Il n'est donc pas possible de savoir quel est le gain d'une telle approche. Néanmoins, l'intégration du courtier dans GridSolve et notamment l'interfaçage avec Matlab valide la preuve de concept.

Les concepts initiaux du projet Grid-TLSE se voulaient génériques et indépendants du domaine d'application. Néanmoins, au fur et à mesure des années (le projet a duré une quinzaine d'années) ils ont perdu de leur généricité. A la fin du projet l'architecture était donc complexe (car pensée pour être générique et ré-utilisable) mais malheureusement spécifique à l'algèbre linéaire creuse, sans pour autant en exploiter toutes les particularités puisque tel n'était pas le but initial. Néanmoins, le projet était fonctionnel et le site accessible en ligne.

CHAPITRE 4

VÉRIFICATION DES COMPOSITIONS DE SERVICES BASÉE SUR LES INTERACTIONS

Ce chapitre s'articule autour de la thématique de la vérification des interactions entre les différents services / composants / pairs d'une composition. La question est alors de savoir si un système composé de pairs communiquant par messages dans le monde asynchrone valide une propriété donnée (par exemple : absence d'interblocage, terminaison, réception de tous les messages, propriété applicative...).

Une façon classique de développer des systèmes distribués est de partir d'une spécification abstraite, tel que l'exclusion mutuelle ou le consensus par exemple, puis d'en dériver une implantation distribuée dans laquelle les variables ne sont pas partagées par différents sites et où des messages / signaux / ports sont utilisés par les sites pour communiquer et interagir. Cette approche remonte à Dijsktra [Dij83], Chandy-Misra avec UNITY [CM88], Back et Kurki-Suonio avec les "action systems" [BK88], Lamport avec TLA [Lam94] et TLA$^+$ [Lam02]. Cela est toujours d'actualité dans la communauté du correct par construction et Event-B [Abr10] est un système bien connu qui adopte cette méthode. À un moment lors du développement, la communication est explicitement introduite pour modéliser l'échange d'information entre sites. Quand le développement est conduit pour la vérification formelle, les propriétés sur la communication sont prouvées suffisantes pour la correction de l'algorithme. Cependant, il est souvent peu clair quelles sont ces propriétés de communication car elles sont généralement noyées au sein de l'algorithme. Il est notamment difficile de remplacer un modèle de communication par un autre sans avoir à refaire complètement la preuve ou le développement.

Ce chapitre propose une approche pour alléger ce travail pour les communications asynchrones point à point. Il s'organise de la façon suivante : dans un premier temps les objectifs et contraintes sont posés, puis les notions de système et de communication asynchrone sont définies.

Ensuite un framework de vérification de compatibilité entre pairs, paramétrable par un modèle de communication, est détaillé. Des modèles de communication asynchrone avec des propriétés d'ordre de délivrance génériques sont ensuite détaillés et leurs liens étudiés. Finalement, des modèles de communication asynchrone avec des propriétés d'ordre de délivrance applicatives sont développés et un mécanisme d'inférence de ces propriétés d'ordre étudié.

Les travaux sont formalisés à l'aide d'Event-B ou de TLA$^+$.

Ces travaux sont les travaux de master et de thèse de Florent Chevrou (Mars 2014 - Novembre 2017), de master de Nathanaël Sensfelder (Mars 2016 - Septembre 2016) et de master d'Adam Shimi (Mars 2017-Septembre 2017). Ces travaux ont été réalisés en collaboration avec Philippe Quéinnec avec qui ces étudiants ont été co-encadrés. Ils ont donné lieu aux publications suivantes : [CHQ16, CHMQ16, CHQ15, CHM$^+$15, SHQ18, SHQ17].

4.1 Objectifs

Lorsque nous nous plaçons dans un monde synchrone, la sémantique des interactions entre deux pairs est bien connue et des outils existent pour décrire les pairs et vérifier leur compatibilité. Utilisons par exemple la notation CCS pour décrire deux pairs. Le premier envoie un message sur le canal a puis un message sur le canal b avant de terminer : $a!.b!.0$. Le second reçoit un message sur le canal a puis un message sur le canal b avant de terminer : $a?.b?.0$. Dans le monde synchrone, lorsque ces deux pairs évoluent en parallèle, un premier rendez-vous a lieu pour l'échange du message sur le canal a, puis un second rendez-vous a lieu pour l'échange du message sur le canal b. Finalement les deux pairs terminent.

Supposons que les interactions entre les pairs soient asynchrones. Comment évolue le système ? Si la communication est FIFO, plusieurs scénarios sont possibles, dont :

1. Le premier pair envoie un message sur le canal a, puis un message sur le canal b puis termine. Puis le second pair reçoit un message sur le canal a, puis sur le canal b puis termine.
2. Le premier pair envoie un message sur le canal a, puis le second pair reçoit un message sur le canal a. Le premier pair envoie un message sur le canal b, puis le second pair reçoit un message sur le canal b. Le premier pair termine puis le second pair termine.

Dans tous les scénarios envisageables le message sur le canal a est reçu avant le message sur le canal b et les deux pairs terminent.

Si la communication est asynchrone sans aucune contrainte d'ordre, les scénarios précédents sont toujours valides, par contre, rien n'empêche le message sur le canal b d'arriver avant le message sur le canal a. Le second pair n'est cependant pas capable de traiter un tel scénario.

Problématique La problématique que nous avons étudiée est la description des systèmes composés de pairs communiquant par message de façon asynchrone.

Objectifs Le premier objectif de ces travaux est de fournir à l'utilisateur des outils formels de vérification de composition de services, dans le monde asynchrone. Pour ce faire, une étude et une formalisation du monde asynchrone sont indispensables. Le second objectif de ces travaux est donc de pouvoir représenter formellement le monde asynchrone dans sa diversité, aussi bien en matière de politique de délivrance (pas de contrainte, canaux FIFO, canaux plus prioritaires...), qu'en multiplicité (pair-à-pair, diffusion, convergence), ou en abstraction (modèles abstraits pour la preuve, modèles concrets pour l'implantation...). Un troisième objectif, qui découle naturellement du premier, est de pouvoir comparer les différents modèles de communication. Pour ce faire, une représentation uniforme des différents modèles est nécessaire, ou au moins la bienvenue, pour faciliter les preuves qui, dans la mesure du possible, doivent être mécanisées.

4.2 Système distribué et communication asynchrone

Un système distribué asynchrone avec communication par message est composé d'un ensemble de pairs qui échangent des messages. Ce chapitre considère la communication point à point où les messages ont un émetteur et au plus un récepteur.

4.2.1 Exécution distribuée

Une exécution distribuée est un ensemble partiellement ordonné d'événements. Comme ce chapitre s'intéresse à la communication asynchrone, les événements sont des événements de communication : événement d'envoi de message, événement de réception de message. L'ordre partiel est habituellement nommé l'ordre causal [Lam78, Mat89, Fid89] et il abstrait les événements indépendants. Les événements se produisent sur des pairs : une fonction étiquette où un événement e s'est produit. En concidérant l'entrelacement d'événements et non la vraie concurrence, un calcul distribué est une extension linéaire d'une exécution distribuée.

Soit `PEER` l'ensemble des pairs, `MESSAGE` l'ensemble d'identificateurs de messages, et `COM` $\triangleq \{Send, Receive\}$ les étiquettes de communication.

Définition 4.2.1 (Exécution distribuée)**.** *Une exécution distribuée* $(E, \prec_c, com, mes, peer)$ *est un ensemble partiellement ordonné avec des fonction d'étiquetage, où* E *est un ensemble énumérable,* $\prec_c$ *est un ordre partiel sur* E*, et* com*,* mes *et* $peer$ *sont des fonctions d'étiquetage de* E *dans* `COM`*,* `MESSAGE` *et* `PEER` *respectivement. Un événement* e *qui se produit sur un pair* $peer(e)$ *est soit l'envoi soit la réception* $(com(e))$ *d'un message* $mes(e)$*.* $(E, \prec_c, com, mes, peer)$ *doit satisfaire :*

- *aucun message n'est envoyé ou reçu plus d'une fois :*

$$\forall e, e' \in E : com(e) = com(e') \wedge mes(e) = mes(e') \Rightarrow e = e'$$

- *un événement de réception est précédé par un événement d'émission :*

$$\forall e \in E : com(e) = Receive \Rightarrow \exists e' \in E : com(e') = Send \wedge mes(e') = mes(e) \wedge e' \prec_c e$$

- *les événements qui se produisent sur le même pair sont totalement ordonnés :*

$$\forall e, e' \in E : peer(e) = peer(e') \Rightarrow e \prec_c e' \vee e' \prec_c e$$

Il faut noter que cette définition autorise les messages envoyés à ne jamais être reçus. Ceci peut être interprété comme la perte d'un message et est indistinguable d'un message restant en transit pour toujours.

Définition 4.2.2 (Calcul distribué)**.** *Soit* $(E, \prec_c, com, mes, peer)$ *une exécution distribuée et* $<_\sigma$ *une extension linéaire de* $\prec_c$ *sur* E*.* $\sigma = (E, \prec_c, <_\sigma, com, mes, peer)$ *est un calcul distribué où* $(E, <_\sigma)$ *est un ensemble totalement ordonné d'événements.*

Occasionnellement un ordre restreint aux événements sur le même pair ($\prec_p$) est utilisé. Cela est simplement réalisé grâce à l'une des deux formulations : $peer(e_1) = peer(e_2) \wedge e_1 \prec_c e_2$ ou $peer(e_1) = peer(e_2) \wedge e_1 <_\sigma e_2$. Comme les événements se produisant sur le même pair sont totalement ordonnés et que $<_\sigma$ contient $\prec_c$, les deux formules sont équivalentes.

L'ensemble des exécutions distribuées est noté `EXEC` et celui des calculs distribués est noté `CALCUL`.

Notations Pour faciliter la lecture, dans la suite un événement e d'une exécution $(E, \prec_c, com, mes, peer)$ ou d'un calcul $(E, \prec_c, <_\sigma, com, mes, peer)$ pourra être écrit sous de la forme :

- $r_p(m)$: réception sur le pair p du message m ($com(e) = Receive$, $mes(e) = m$ et $peer(e) = p$) ;
- $s_p(m)$: émission sur le pair p du message m ($com(e) = Send$, $mes(e) = m$ et $peer(e) = p$).

4.2.2 Des événements aux exécutions distribuées

Les calculs distribués et la communication asynchrone point à point ont été formalisés à l'aide d'Event-B. Une description abstraite a été raffinée jusqu'à introduire les concepts définis précédemment. L'architecture générale des raffinements initiaux est exposée dans la Figure 4.1. Les machines initiales introduisent les événements et établissent le lien entre deux événements : une cause et une conséquence (contexte `Event` et machines `Events` et `Communication`). Les pairs sont ensuite introduits ainsi que la localisation des événements sur les pairs (machine `DistributedExecution` et contexte `Peers`). Cela définit les exécutions distribuées où les événements sont causalement liés (ordre partiel $\prec_c$). Ensuite, un calcul distribué (run) est défini à partir d'une extension linéaire de $\prec_c$ où les événements sont totalement ordonnés selon

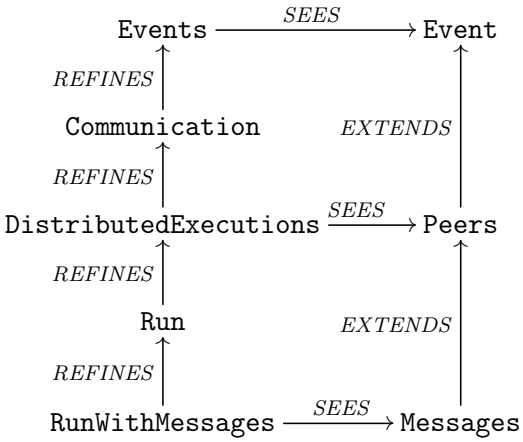


FIGURE 4.1 – Architecture générale des machines et contextes Event-B

$<_\sigma$ (machine Run). Finalement les messages sont introduits (contexte Messages et machine RunWithMessages). Les messages établissent le lien entre deux événements : un événement d'envoi et son événement de réception correspondant. Cette dernière machine est conforme aux définitions 4.2.1 and 4.2.2 d'exécutions et calculs distribués. Cela correspond à un système distribué avec communication asynchrone point à point.

Tous les modèles discutés dans ce chapitre sont disponibles en ligne [1] : http://hurault.perso.enseeiht.fr/MenagerieOfRefinements/. Cette page donne également les indications pour rejouer les preuves (principalement les versions de Rodin et des prouveurs SMT à utiliser). La suite de cette section est dédiée à l'étude détaillée de ces machines.

Attention, l'utilisation d'Event-B entraîne un souci de vocabulaire : le terme évènement est à la fois utilisé pour les évènements Event-B et pour les évènements d'envoi et de réception de messages. Les premiers manipulant les seconds, le discours est parfois acrobatique.

4.2.2.1 Événements

Le contexte Event introduit l'ensemble EVENT. La machine Events acte simplement que de nouveaux événements peuvent se produire. La variable past enregistre les événements passés pour s'assurer qu'un événement ne peut pas se produire deux fois. La figure 4.2 donne le code de cette machine.

4.2.2.2 Liens entre événements

La machine Communication introduit la notion de lien entre un événement et sa cause. Ce lien sera plus tard réalisé grâce à un message où l'envoi de celui-ci sera une cause et la réception

1. Les noms des modèles sont sous la forme L_Name, où le préfixe est utilisé pour que les modèles soient affichés dans leur ordre de raffinement sous Rodin. Le texte du chapitre omet souvent la lettre en préfixe mais les extraits des modèles la conservent pour référence.

```
MACHINE Events
SEES Event
VARIABLES
   past   // Set of events that have happened
INVARIANTS
  Tpast: past ∈ ℙ(EVENT)

EVENT INITIALISATION THEN past := ∅

EVENT happen
   ANY e // New event
   WHERE
      grd1: e ∈ EVENT \ past
   THEN
      act1:    past := past ∪ {e}
 END
```

FIGURE 4.2 – Machine Events

```
INVARIANTS
   Tlinks: links ∈ past ↔past
   inv0:   dom(links) ∩ ran( links ) = ∅
    // Causes (emissions) and consequences (receptions) are distinct
   inv1:   ∀ e1, e2, c · e1 ↦ c ∈ links ∧ e2 ↦ c ∈ links ⇒ e1 = e2
    // a consequence has only one (direct) cause
   inv2:   ∀ e, c1, c2 · e ↦ c1 ∈ links ∧ e ↦ c2 ∈ links ⇒ c1 = c2
    // a cause has at most one consequence
```

FIGURE 4.3 – Invariants de la machine Communication

une conséquence. Une nouvelle variable links est introduite. La communication étant point à point les invariants attendus sont : la différentiation des causes (réceptions) et des conséquences (émissions), l'unicité d'une cause pour une conséquence donnée et le fait qu'une cause a au plus une conséquence. La figure 4.3 fournit le code Event-B des invariants.

L'événement happen est raffiné en deux nouveaux événements : un événement create qui est exactement happen et un nouvel événement link qui établit un lien. create génère un évènement autonome (future émission), alors que link génère un évènement lié à une cause (future réception). Un extrait de la machine Event-B est donné dans la figure 4.4.

4.2.2.3 Exécutions distribuées

Le contexte Peer étend Event avec PEER, un ensemble d'identifiants de pair. La machine DistributedExecution ajoute deux variables : peerOf qui associe à chaque événement le pair sur lequel il s'est produit, et prec qui est la relation de causalité entre événements. L'expression Event-B e1↦e2 ∈ prec est aussi notée $e1 <_c e2$ dans le texte. Les principaux invariants indiquent que prec est un ordre partiel, est totalement ordonné sur les pairs, et contient links.

Les invariants de la machine sont donnés dans la figure 4.5 et un extrait de ses évènements

```
MACHINE Communication
REFINES Events
...
EVENT create REFINES happen extended
  ...   // unchanged

EVENT link REFINES happen // A new event occurs, with a link to its cause
  ANY e        // New event
      cause    // Cause event
  WHERE
     grd1:   e  ∈ EVENT \ past
     grd2:   cause  ∈ past          // The cause event has already happened
     grd3:   cause  ∉ dom(links)    // A cause serves once only
     grd4:   cause  ∉ ran( links )  // A cause is not a consequence
  THEN
     act1:   past := past ∪ {e}
     act2:   links :|  links ' = links ∪ {cause ↦ e}
END
```

FIGURE 4.4 – Extrait de la machine `Communication`

```
INVARIANTS
  Tprec: prec  ∈ past ↔past
  TpeerOf: peerOf  ∈ past →PEER
  inv1: (past ◁ id ) ⊆ prec // prec is reflexive
  inv2: prec ; prec ⊆ prec // prec is transitive
  inv3: prec ∩ prec^−1 ⊆id // prec is anti−symmetric
  inv4: links ⊆ prec // prec contains (the reflexive transitive closure of) links
  inv5: ∀ e1, e2 · e1 ∈ past ∧ e2 ∈ past ∧ peerOf(e1) = peerOf(e2) ⇒ e1 ↦ e2 ∈ prec ∨ e2 ↦ e1 ∈ prec
            // Events occurring on the same peer are totally ordered
```

FIGURE 4.5 – Invariants de la machine `DistributedExecution`

dans la figure 4.6. L'événement `create` est similaire à `link`, sans la quatrième ligne de `+act4` qui correspond à la fermeture transitive de `prec` vis à vis de la `cause`. Le choix a été fait de construire explicitement la fermeture transitive. Il aurait aussi été possible de construire une relation non transitive et de définir `prec` comme la fermeture transitive de celle-ci. Or dans Rodin, il est assez compliqué de manipuler les fermetures transitives. C'est donc pour une facilité de réalisation des preuves que le choix de construire explicitement la fermeture transitive a été fait.

4.2.2.4 Calculs distribués

Le raffinement suivant est la machine `Run` qui introduit les calculs distribués grâce à une variable `run` qui ordonne totalement les événements. L'expression Event-B `e1↦e2 ∈ run` est aussi notée $e1 <_\sigma e2$ dans le texte. Ses invariants sont donnés dans la figure 4.7. La variable `run` est construite par superposition dans les actions `create` et `link` comme montré dans la figure 4.8.

```
MACHINE DistributedExecution
REFINES Communication
...
EVENT link REFINES link extended
   ANY
       ...
       p
   WHERE
       ...
       +grd5: p ∈ PEER
   THEN
       ...
       +act3: peerOf := peerOf ∪ {e ↦ p}
              // The new event is causally after all events from the same peer (third line)
              // and after all events that causally precedes the cause (fourth line)
       +act4: prec := prec
                    ∪ {e ↦ e}
                    ∪ {ep · ep ∈ past ∧ (∃ ep2 · ep2 ∈ past ∧ peerOf(ep2) = p ∧ ep ↦ ep2 ∈ prec) | ep ↦ e}
                    ∪ {ep · ep ∈ past ∧ ep ↦ cause ∈ prec | ep ↦ e}
END
```

FIGURE 4.6 – Extrait de la machine `DistributedExecution`

```
INVARIANTS
  Trun: run ∈ past ⇸ past
  inv0: ∀ e1, e2 · e1 ∈ past ∧ e2 ∈ past ⇒ e1 ↦ e2 ∈ run ∨ e2 ↦ e1 ∈ run // run is total
  inv1: (past ◁ id) ⊆ run // run is reflexive (consequence of inv0)
  inv2: run ; run ⊆ run // run is transitive
  inv3: run ∩ run^-1 ⊆ id // run is anti−symmetric
  inv4: prec ⊆ run // run extends prec
  inv5: ∀ e1, e2 · e1 ∈ past ∧ e2 ∈ past ∧ peerOf(e1) = peerOf(e2) ⇒ ((e1 ↦ e2 ∈ prec) ⇔ (e1 ↦ e2 ∈ run))
```

FIGURE 4.7 – Invariants de la machine `Run`

```
MACHINE Run
REFINES DistributedExecution
...
EVENT create REFINES create extended
  ...
  +act5: run := run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}

EVENT link REFINES link extended
  ...
  +act5: run := run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}
```

FIGURE 4.8 – Extrait de la machine `Run`

4.2.2.5 Messages

Finalement, les messages sont introduits. Un message sert de lien entre son événement d'envoi et son événement de réception. La machine `RunWithMessages` ajoute deux variables `mesOf` et `comOf` qui, pour chaque événement, spécifie le message et le type d'événement (envoi ou réception). Inversement, la variable `links` et le paramètre `cause` de l'événement `link` (raffiné en `receive`) sont supprimés. Un invariant de liaison `Glinks` et un témoin adéquat pour `receive` sont fournis. Cette machine (figure 4.9) modélise une exécution distribuée pour la communication asynchrone point à point.

4.2.2.6 Discussion

Le modèle initial peut paraître artificiel avec son unique invariant de typage et son unique événement trivial. Il fournit cependant la base pour le raffinement, qui à chaque étape, introduit un élément essentiel jusqu'à l'obtention d'une modélisation d'une exécution distribuée. La machine `Communication` introduit la communication point à point grâce à une relation entre deux événements. En jouant avec les gardes, d'autres modèles peuvent être spécifiés : imposer que les conséquences soient aussi les causes (communication synchrone) ; autoriser plusieurs conséquences pour une cause (diffusion) ; autoriser plusieurs causes pour une conséquence (convergence). La machine suivante, `DistributedExecution`, introduit les pairs et la relation de causalité. Puis la machine `Run` introduit une observation d'une exécution distribuée comme un ordre total sur les événements. Finalement `RunWithMessages` identifie la relation entre deux événements grâce à un message et exprime deux invariants essentiels sur les messages (unicité des réceptions et réception après émission). Cette dernière machine modélise un modèle de communication purement asynchrone avec de la communication par message point à point.

4.2.3 Modèles de communication

La dernière machine obtenue peut ensuite être raffinée pour imposer plus de contraintes sur la communication, c'est-à-dire contraindre la réception et/ou l'envoi des messages. Ces contraintes peuvent être génériques (l'ordre de réception des messages dépend de l'ordre d'émission de ceux-ci) ou applicatives (pour une application donnée, un message est défini plus prioritaire qu'un autre).

4.2.3.1 Modèles de communication génériques

Dans un premier temps, l'intuition des modèle est donnée. Ils sont formellement définis dans la section 4.4.

RSC Realizable with Synchronous Communication [CBMT96, KS11]. Entre un événement d'envoi d'un message et l'événement de réception de ce même message, il ne peut pas y avoir d'événement de communication. C'est-à-dire que si un message est reçu, l'événement

MACHINE RunWithMessages
REFINES Run
SEES Messages
VARIABLES
past
peerOf
prec
run
mesOf // *Message mesof(e) of a communication event e*
comOf // *label comOf(e) (Send or Receive) of a communication event e*
INVARIANTS
TcomOf: comOf ∈ past →COM
TmesOf: mesOf ∈ past →MESSAGE
inv1: ∀ e1, e2 · e1 ∈ past ∧ e2 ∈ past ∧ comOf(e1) = comOf(e2) ∧mesOf(e1) = mesOf(e2) ⇒e1 = e2
// *no message is sent or received more than once*
inv2: ∀ e · e ∈ past ∧ comOf(e) = Receive ⇒(∃ es · es ∈ past ∧ comOf(es) = Send ∧ mesOf(e) = mesOf(es) ∧es ↦ e ∈ prec)
// *a receive event is preceded by a send event*
Glinks: ∀ es, er · es ↦er ∈ links ⇒ comOf(es) = Send ∧comOf(er) = Receive ∧mesOf(es) = mesOf(er)

EVENT send **REFINES** create **extended**
ANY e // *New event*
p // *Peer where the event occurs*
m // *Sent message*
WHERE
...
grd3: m ∈ MESSAGE \ran(mesOf)
THEN
...
+act5: mesOf := mesOf ∪{e ↦ m}
+act6: comOf := comOf ∪{e ↦ Send}

EVENT receive **REFINES** link
ANY e // *New event*
p // *Receiver*
m // *Received message*
WHERE
grd1: e ∈ EVENT \ past
grd4: p ∈ PEER
grd5: m ∈ MESSAGE
grd6: ∀ ep · ep ∈ past ∧ comOf(ep) = Receive ⇒mesOf(ep) ≠ m // *m has not already been received*
grd7: ∃ es · es ∈ past ∧ comOf(es) = Send ∧mesOf(es) = m // *m has been sent*
WITH
cause: cause ∈ past ∧ comOf(cause) = Send ∧mesOf(cause) = m
links ': links ' = links ∪ {es · es ∈ past ∧ comOf(es) = Send ∧mesOf(es) = m | es ↦ e}
THEN
act1: past := past ∪ {e}
act3: peerOf := peerOf ∪ {e ↦ p}
act4: prec := prec ∪ {e ↦ e}
∪ {ep · ep ∈ past ∧ (∃ ep2 · ep2 ∈ past ∧ peerOf(ep2) = p ∧ep ↦ ep2 ∈ prec) | ep ↦ e}
∪ {ep · ep ∈ past ∧ (∃ es · es ∈ past ∧ comOf(es) = Send ∧mesOf(es) = m ∧ep ↦ es ∈ prec) | ep ↦ e}
act5: run := run ∪ {e ↦ e} ∪ {ep · ep ∈ past | ep ↦ e}
+act6: mesOf := mesOf ∪{e ↦ m}
+act7: comOf := comOf ∪{e ↦ Receive}

FIGURE 4.9 – Extrait de la machine `RunWithMessages`

d'envoi d'un message est immédiatement suivi par l'événement de réception de celui-ci (dans le cas d'absence d'événements internes). Si le couple ⟨envoi, réception⟩ est vu atomiquement, cela correspond à de la communication synchrone.

Interprétation orientée structure de données : les messages sont stockés dans un buffer de taille 1.

FIFO n-n Les messages sont globalement ordonnés et sont délivrés dans leur ordre d'émission. Ce modèle est souvent utilisé comme première étape pour passer de la communication synchrone à la communication asynchrone en découplant l'événement d'envoi et celui de réception.

Interprétation orientée structure de données : les messages sont stockés dans une unique file où tous les émetteurs déposent les messages et où tous les récepteurs récupèrent les messages.

FIFO 1-n Les messages envoyés par un même pair sont reçus dans leur ordre d'émission.

Interprétation orientée structure de données : les messages sont stockés dans n files (une par émetteur) où chaque émetteur dépose ses messages et où tous les récepteurs récupèrent les messages.

FIFO n-1 Les messages à destination d'un même pair sont reçus dans leur ordre d'émission.

Interprétation orientée structure de données : les messages sont stockés dans n files (une boite aux lettres par récepteur). Les émetteurs déposent les messages dans la boite aux lettres du destinataire, ils sont alors reçus dans leur ordre de dépôt.

FIFO 1-1 Les messages entre un couple de pairs sont reçus dans leur ordre d'émission. C'est le modèle classiquement appelé FIFO dans la littérature.

Interprétation orientée structure de données : les messages sont stockés dans n^2 files (une file par couple émetteur / récepteur).

Causal Les messages sont reçus en accord avec la causalité de leur émission [Lam78]. Si un message m_1 est causalement émis avant un message m_2 (i.e. il existe un chemin causal entre l'émission du premier et celle du second), alors un pair ne peut pas recevoir m_2 puis m_1.

Async Aucun ordre n'est imposé sur la réception des messages. La machine `RunWithMessages` de la section précédente correspond à ce modèle.

Interprétation orientée structure de données : les messages sont stockés dans un ensemble d'où ils sont extraits dans n'importe quel ordre.

4.2.3.2 Modèles de communication applicatifs

Dans un premier temps, l'intuition de ces modèles est donnée. Ils sont formellement définis dans la section 4.5.

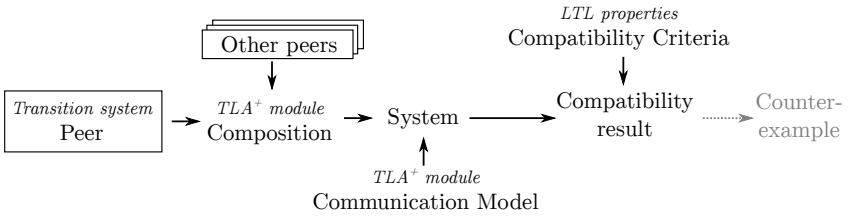


FIGURE 4.10 – Architecture du framework

L'idée est ici de pouvoir donner la priorité à un message sur un autre. La condition de délivrance ne se fait plus en fonction de l'ordre d'émission des messages, mais en fonction des autres messages en transit et de leur priorité. Par exemple, il peut être intéressant de traiter en priorité des messages d'exceptions. Pour faciliter l'expression de ces priorités, les messages sont étiquetés (nous parlerons dans la suite de canaux). C'est sur les étiquettes de messages que sont données les priorités. Ces priorités forment un ensemble de contraintes statiques qui force des réceptions à être désactivées en fonction de l'état de réseau. (B BLOCKS c) signifie que si au moins un message est en transit étiqueté par un élément de B, alors la réception d'un message étiqueté par c est *désactivée*.

4.3 Le framework pour la vérification de compatibilité

L'objectif est maintenant de vérifier si une composition de pairs (dont une description du comportement est fournie) valide une propriété en fonction du modèle de communication gérant les interactions entre les pairs. Une hypothèse forte de ce travail est que ce n'est pas le pair qui récupère les messages sur le réseau, mais c'est le modèle de communication qui délivre les messages aux pairs. Prenons l'exemple d'un client / serveur. Le client peut se connecter au serveur en envoyant son login puis son mot de passe. Le serveur attend deux messages, un message de login, suivi d'un message de mot de passe. Or si le modèle de communication asynchrone sous-jacent est asynchrone pur (aucune contrainte d'ordre imposée), le mot de passe est susceptible d'arriver avant le login. Le serveur doit donc être capable de traiter ce cas de figure, sinon une erreur doit être détectée. Par contre, si le modèle de communication sous-jacent est FIFO 1-1, le serveur n'aura pas à prendre en compte cette possibilité d'inversion des messages.

4.3.1 Aperçu du framework

La figure 4.10 donne une vue globale du framework réalisé en TLA$^+$. Il est détaillé dans [Che17] et testable en ligne à l'adresse suivante : http://vacs.enseeiht.fr

Le framework est basé sur une description des pairs et du modèle de communication à l'aide de systèmes de transitions. Les messages n'ont pas de destinataires explicites mais sont envoyés sur des canaux ce qui facilite la description des pairs. Notamment, les pairs peuvent être décrits

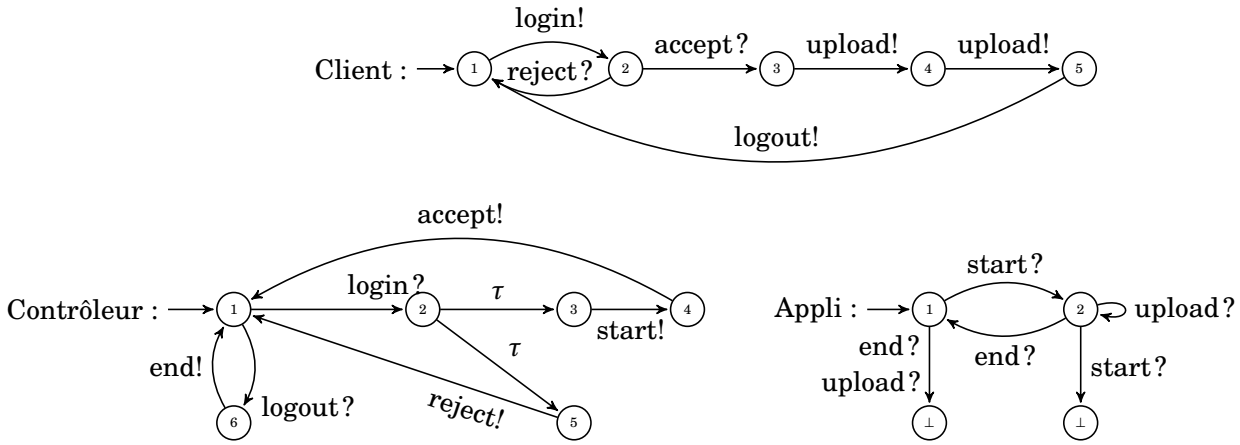


FIGURE 4.11 – Un système client-serveur

en CCS et le framework construit alors automatiquement le système de transition. Le modèle de communication délivrant les messages aux pairs, il a besoin de connaître les canaux écoutés par les pairs, pour ne leur délivrer que des messages pertinents. Le terme "intérêt des pairs" est utilisé dans la suite pour représenter l'ensemble des canaux écoutés par un pair dans un état donné.

Le système évolue selon une variante du produit synchronisé entre les pairs et le modèle de communication : le modèle de communication et un pair évoluent sur l'envoi ou la réception d'un message, pendant que les autres pairs restent dans le même état.

Le but est de vérifier que le système composé de pairs valide une (ou plusieurs) propriété(s). Ces propriétés sont exprimées en logique LTL [MP92] et peuvent être génériques (tous les pairs terminent, il n'y a pas de réception inattendue. . .) ou applicatives.

Les pairs, les modèles de communication et la glue nécessaire à l'évolution du système sont décrits par des modules TLA$^+$. La propriété LTL est vérifiée à l'aide du vérificateur de modèles TLC.

4.3.2 Exemple

Prenons l'exemple du client-serveur disponible sur le démonstrateur en ligne et illustré par la figure 4.11. Le serveur est décomposé en un contrôleur et une application. Le contrôleur accepte ou rejette le client et démarre / stoppe l'application quand nécessaire. Le client fait une demande de connexion, s'il est rejeté, il peut demander à nouveau. S'il est accepté, il fait deux *upload* puis se déconnecte. L'application accepte un message de démarrage, puis un nombre quelconque d'upload, avant de revenir dans l'état initial à la réception d'un message d'arrêt. Si elle reçoit un message d'arrêt ou d'upload alors qu'elle n'a pas démarré, cela provoque une erreur (état $\perp$). De même si elle reçoit un message de démarrage alors qu'elle tourne déjà, cela provoque une erreur.

Avec un modèle purement asynchrone, le framework fait remonter une erreur :

```
Client: 1 --[login!]--> 2
Contro: 1 --[login?]--> 2
Contro: 2 --[tau]--> 3
Contro: 3 --[start!]--> 4
Contro: 4 --[accept!]--> 1
Client: 2 --[accept?]--> 3
Client: 3 --[upload!]--> 4
Appli : 1 --[upload?]--> FAILURE
```

En effet, il n'y a aucune garantie que le message *start* arrive avant le message *upload*, même si c'est le message *accept*, envoyé avant le message *start* qui déclenche l'envoi du message *upload*.

Pour que ce cas de figure ne se produise pas, il faut, par exemple, que le modèle de communication soit *Causal* ou que des contraintes applicatives assurent que l'application ne soit pas trop en retard (voir section 4.5.3.2).

4.3.3 Les contenus applicatifs

Dans le cadre du projet ANR PARDI [2] le framework a été étendu pour ajouter du contenu applicatif aux messages. En effet dans les travaux développés ici, les messages sont des évènements sans contenu. Dans le but de vérifier des algorithmes distribués réalistes, il est important que les messages puissent transporter de l'information. Cette extension a été ajoutée au framework.

La seule difficulté a été le codage de la réception. Dû à la nature de TLA$^+$, il fallait un quantificateur existentiel sur la donnée reçue. Dans le cas d'un domaine de donnée infini, TLC n'est alors plus capable de vérifier le système. Une variable a donc été ajoutée pour mémoriser les données envoyées mais pas encore reçues, ce qui permet une quantification existentielle sur un ensemble fini.

4.4 Modèles de communication génériques

Nous venons de présenter un framework qui permet de vérifier la compatibilité de pairs modulo un modèle de communication. Différents modèles de communication peuvent être définis. Dans cette section nous allons revenir sur les sept modèles de communication asynchrone point à point génériques dont l'intuition a été donnée précédemment en les définissant formellement et en étudiant leurs liens.

2. http://pardi.enseeiht.fr

4.4.1 Les modèles

Les études de ces modèles ont été menées en TLA$^+$ et en Event-B. Selon les travaux, les modèles diffèrent légèrement : dans les travaux en Event-B les messages possèdent un destinataire explicite alors que dans les travaux en TLA$^+$ les messages sont envoyés sur des canaux et n'importe quel pair écoutant ce canal peut recevoir le message. Ce sont les modèles TLA$^+$ qui ont été intégrés au framework défini précédemment.

Cette différence s'explique par une légère différence d'objectifs des travaux. Ceux menés avec Event-B ont pour unique but l'étude des modèles de communication à proprement parler. Ceux menés avec TLA$^+$ sont intégrés au framework qui permet de vérifier la compatibilité de pairs modulo un modèle de communication. La présence de ces pairs nous a motivé à ne pas mettre de destinataire explicite pour être plus expressif dans la description des pairs.

Une autre légère différence est que les modèles TLA$^+$ possèdent une action interne (τ) alors qu'elle n'a pas été prise en compte dans les modèles Event-B. Les modèles Event-B ont été définis dans le but d'étudier la communication en isolation. Donc seuls les événements de communication importent. Les modèles TLA$^+$ étant intégrés au framework, il est, encore une fois, nécessaire d'avoir une modélisation plus expressive, et donc de laisser la possibilité aux pairs et aux modèles de communication de réaliser des actions internes.

4.4.1.1 Définitions basées sur les ordres

Comme explicité par l'intuition des modèles de communication, ceux-ci imposent un ordre de réception des messages en respectant un prédicat spécifique, qui fait, entre autre, intervenir les ordres d'émission des messages.

Définition des modèles Un calcul distribué σ est conforme à un modèle de communication CM si toutes ses réceptions sont correctement ordonnées vis-à-vis du prédicat de délivrance du modèle. Sauf pour le modèle RSC, l'ensemble des calculs distribués valides pour CM est défini selon un schéma commun de la forme :

$$\mathscr{C}alcul(CM) \triangleq \left\{ \sigma \in \texttt{CALCUL} \;\middle|\; \begin{array}{l} \forall m_1, m_2 \in \texttt{MESSAGE}, \forall p_{r_1}, p_{r_2}, p_{s_1}, p_{s_2} \in \texttt{PEER} : \\ \quad r_{p_{r_1}}(m_1) \in \sigma \wedge r_{p_{r_2}}(m_2) \in \sigma \\ \quad \wedge \;\text{condition sur } s_{p_{s_1}}(m_1),\, s_{p_{s_2}}(m_2),\, r_{p_{r_1}}(m_1),\, r_{p_{r_2}}(m_2) \\ \quad \Rightarrow r_{p_{r_1}}(m_1) \prec_\sigma r_{p_{r_2}}(m_2) \end{array} \right\}$$

La condition de délivrance de chacun des modèles est donnée dans le tableau 4.1. Par exemple pour le modèle FIFO 1-1, dont la définition informelle dit que les messages entre un couple de pairs sont reçus dans leur ordre d'émission, la condition de délivrance spécifie que si les messages m_1 et m_2 sont reçus ($r_{p_{r_1}}(m_1) \in \sigma \wedge r_{p_{r_2}}(m_2) \in \sigma$), que m_1 a été envoyé avant

$FIFO\,n-n$	$FIFO\,n-1$	$Causal$	$FIFO\,1-1$	$FIFO\,1-n$
$s_{p_{s_1}}(m_1) \prec_\sigma s_{p_{s_2}}(m_2)$	$s_{p_{s_1}}(m_1) \prec_\sigma s_{p_{s_2}}(m_2)$ $\wedge p_{r_1} = p_{r_2}$	$s_{p_{s_1}}(m_1) \prec_c s_{p_{s_2}}(m_2)$ $\wedge p_{r_1} = p_{r_2}$	$s_{p_{s_1}}(m_1) \prec_c s_{p_{s_2}}(m_2)$ $\wedge p_{r_1} = p_{r_2}$ $\wedge p_{s_1} = p_{s_2}$	$s_{p_{s_1}}(m_1) \prec_c s_{p_{s_2}}(m_2)$ $\wedge p_{s_1} = p_{s_2}$

TABLE 4.1 – Prédicats de délivrance des différents modèles de communication

m_2 ($s_{p_{s_1}}(m_1) \prec_c s_{p_{s_2}}(m_2)$), depuis le même pair ($p_{s_1} = p_{s_2}$) et qu'ils sont reçus sur le même pair ($p_{r_1} = p_{r_2}$), alors ils doivent être reçus dans leur ordre d'envoi ($r_{p_{r_1}}(m_1) \prec_\sigma r_{p_{r_2}}(m_2)$).

La condition de délivrance du modèle RSC [3] est, quant à elle, de la forme :

$$\mathscr{C}alcul(RSC) \triangleq \left\{ \sigma \in \texttt{CALCUL} \;\middle|\; \begin{array}{l} \forall m \in \texttt{MESSAGE}, \forall p_r, p_s \in \texttt{PEER} : \\ r_{p_r}(m) \in \sigma \Rightarrow \forall e \in \sigma : s_{p_s}(m) \prec_\sigma e \prec_\sigma r_{p_r}(m) \Rightarrow e \in \{r_{p_r}(m), s_{p_s}(m)\} \end{array} \right\}$$

Notons que ces définitions autorise un message à ne jamais être reçu.

Un exemple de modèle Event-B Chacun des sept modèles a été décrit en Event-B. Ils sont caractérisés par un invariant qui décrit les propriétés d'ordre définies précédemment. Par exemple, l'invariant de la machine correspondant au modèle FIFO 1-1 est le suivant :

```
∀ es1, er1, es2, er2 · es1 ∈ past ∧ er1 ∈ past ∧ es2 ∈ past ∧ er2 ∈ past
    ∧ comOf(es1) = Send ∧comOf(es2) = Send ∧comOf(er1) = Receive ∧comOf(er2) = Receive
    ∧ mesOf(es1) = mesOf(er1) ∧mesOf(es2) = mesOf(er2)
    ∧ peerOf(es1) = peerOf(es2) ∧ peerOf(er1) = peerOf(er2)
    ∧ es1 ↦ es2 ∈ prec
    ⇒ er1 ↦ er2 ∈ run
```

Cet invariant dit que si $es1$ et $es2$ sont des évènements d'envoi ($comOf(es1) = Send \wedge comOf(es2) = Send$), $er1$ et $er2$ sont des évènements de réception ($comOf(er1) = Receive \wedge comOf(er2) = Receive$), que $er1$ (resp. $er2$) est la réception associée à $es1$ (resp. $es2$) ($mesOf(es1) = mesOf(er1) \wedge mesOf(es2) = mesOf(er2)$), que les émissions ont lieu sur le même pair ($peerOf(es1) = peerOf(es2)$), que les réceptions ont lieu sur le même pair ($peerOf(er1) = peerOf(er2)$), que $es1$ a eu lieu avant $es2$ ($es1 \rightarrow es2 \in prec$) alors les réceptions ont lieu dans le même ordre ($er1 \rightarrow er2 \in run$).

Un extrait de la machine est présentée dans la figure 4.12. Elle raffine la machine $RunWithMessages$ qui, pour rappel, correspond aux calculs distribués avec communication asynchrone point à point par messages.

Toutes les machines ont été écrites sur le même modèle. Un invariant représente la condition de délivrance. Les modèles devant permettre autant d'exécutions que cet invariant l'autorise, la

3. Le modèle original donnait une propriété sur les exécutions distribuées et non sur les calculs. Pour qu'une exécution soit valide dans le modèle RSC original, il faut qu'une exécution causalement équivalente valide la propriété énoncée ici.

MACHINE Fifo11Event
REFINES RunWithMessages
SEES MESSAGES
VARIABLES *// unchanged*
past
peerOf
prec
run
mesOf
comOf

EVENT send **REFINES** send **extended**
ANY e
p
m
WHERE
grd1: $e \in EVENT \setminus past$
grd2: $p \in PEER$
grd3: $m \in MESSAGE \setminus ran(mesOf)$
// weakest precondition of the fifo11 ordering invariant
ordering: $\forall es1, er1, es2, er2 \cdot es1 \in past \cup \{e\} \wedge er1 \in past \cup \{e\} \wedge es2 \in past \cup \{e\} \wedge er2 \in past \cup \{e\}$
$\wedge (comOf \cup \{e \mapsto Send\})(es1) = Send \wedge (comOf \cup \{e \mapsto Send\})(es2) = Send$
$\wedge (comOf \cup \{e \mapsto Send\})(er1) = Receive \wedge (comOf \cup \{e \mapsto Send\})(er2) = Receive$
$\wedge (mesOf \cup \{e \mapsto m\})(es1) = (mesOf \cup \{e \mapsto m\})(er1) \wedge (mesOf \cup \{e \mapsto m\})(es2) = (mesOf \cup \{e \mapsto m\})(er2)$
$\wedge (peerOf \cup \{e \mapsto p\})(es1) = (peerOf \cup \{e \mapsto p\})(es2) \wedge (peerOf \cup \{e \mapsto p\})(er1) = (peerOf \cup \{e \mapsto p\})(er2)$
$\wedge es1 \mapsto es2 \in run \cup \{e \mapsto e\} \cup \{ep \cdot ep \in past \mid ep \mapsto e\}$
$\Rightarrow er1 \mapsto er2 \in run \cup \{e \mapsto e\} \cup \{ep \cdot ep \in past \mid ep \mapsto e\}$
THEN *// unchanged*
act1: $past := past \cup \{e\}$
act2: $peerOf := peerOf \cup \{e \mapsto p\}$
act3: $prec := prec \cup \ldots$
act4: $run := run \cup \ldots$
act5: $mesOf := mesOf \cup \{e \mapsto m\}$
act6: $comOf := comOf \cup \{e \mapsto Send\}$

EVENT receive **REFINES** receive **extended**
...
WHERE
...
// weakest precondition of the fifo11 ordering invariant
ordering: $\forall es1, er1, es2, er2 \cdot es1 \in past \cup \{e\} \wedge er1 \in past \cup \{e\} \wedge es2 \in past \cup \{e\} \wedge er2 \in past \cup \{e\}$
$\wedge (comOf \cup \{e \mapsto Receive\})(es1) = Send \wedge (comOf \cup \{e \mapsto Receive\})(es2) = Send$
$\wedge (comOf \cup \{e \mapsto Receive\})(er1) = Receive \wedge (comOf \cup \{e \mapsto Receive\})(er2) = Receive$
$\wedge (mesOf \cup \{e \mapsto m\})(es1) = (mesOf \cup \{e \mapsto m\})(er1) \wedge (mesOf \cup \{e \mapsto m\})(es2) = (mesOf \cup \{e \mapsto m\})(er2)$
$\wedge (peerOf \cup \{e \mapsto p\})(es1) = (peerOf \cup \{e \mapsto p\})(es2) \wedge (peerOf \cup \{e \mapsto p\})(er1) = (peerOf \cup \{e \mapsto p\})(er2)$
$\wedge es1 \mapsto es2 \in run \cup \{e \mapsto e\} \cup \{ep \cdot ep \in past \mid ep \mapsto e\}$
$\Rightarrow er1 \mapsto er2 \in run \cup \{e \mapsto e\} \cup \{ep \cdot ep \in past \mid ep \mapsto e\}$
THEN
... *// unchanged*

FIGURE 4.12 – Modèle de communication FIFO 1-1 décrit avec des événements. Les préconditions les plus faibles sont utilisées pour s'assurer que les événements ne sont pas plus contraints que l'invariant d'ordre qui définit le modèle.

précondition la plus faible (weakest precondition) de cet invariant a été choisie comme garde des évènements (Event-B) d'envoi et de réception.

4.4.1.2 Définitions basées sur des histoires

La formalisation précédente des modèles est abstraite. Une seconde formalisation unifiée des modèles a été définie. Elle est basée sur des histoires de messages. Cette formalisation repose sur :

- `NET` : le réseau qui contient l'ensemble des messages en transit (i.e. envoyés mais non encore reçus) ;
- `HG` : l'histoire globale qui contient l'ensemble des messages ayant été envoyés ;
- `HC` : pour chaque pair, l'histoire causale de celui-ci ;
- `HL` : pour chaque pair, l'histoire locale de celui-ci.

Intuitivement, `HG` permet de reconstruire $\prec_\sigma$ et `HC` permet de reconstruire $\prec_c$. `HL` permet de reconstruire l'ordre local. Comme dit précédemment, celui-ci peut être retrouvé à partir de l'ordre global ou causal. Nous aurions donc pu nous passer de `HL`. C'est d'ailleurs le cas dans les modèles Event-B. Mais pour des facilités d'écriture, les histoires locales sont utilisées dans les modèles TLA$^+$.

Les messages en transit sont de la forme :

- $\langle id, chan, peer, HL[peer], HC[peer], HG\rangle$ pour les modèles TLA$^+$ où le destinataire n'est pas explicite. Le message est composé d'un identifiant, du canal sur lequel il est émis, du pair émetteur et des histoires locale, causale et globale ;
- $\langle peer, dest, HC[peer], HG\rangle$ pour les modèles Event-B où le destinataire est explicite. Les messages ne sont en pratique pas codés avec des quadruplets, mais à l'aide de quatre fonctions d'étiquetage : $origOf$ (émetteur du message), $destOf$ (destinataire du message), $hcOf$ (histoire causale du message, i.e. l'histoire causale du pair d'émission au moment de l'envoi), $hgOf$ (histoire globale du message, i.e. l'histoire globale du pair d'émission au moment de l'envoi).

Définition des modèles Tous les modèles sont définis sur le même schéma. Lors de l'envoi d'un message, celui-ci est construit (selon la forme explicitée précédemment), ajouté au réseau et les histoires globale, causale et locale mises à jour avec l'ajout du message dans celles-ci. Lors de la réception d'un message, celui-ci est retiré du réseau et l'histoire causale du pair de réception est mise à jour avec l'ajout à celle-ci de l'histoire causale du message.

Seules changent les gardes des actions / événements d'émission et réception. Le tableau 4.2 résume les gardes des actions pour les modèles TLA$^+$. Le tableau 4.3 résume les gardes des événements pour les modèles Event-B. Il faut noter que les paramètres des actions / événements ne sont pas les mêmes selon que nous ayons des destinataires explicites ou non.

Modèle	send(p,c)	receive(p,c,L)
RSC	$\texttt{NET} = \varnothing$	$\exists m = \langle \_, c_m, \_, \_, \_, \_ \rangle \in \texttt{NET} : c_m = c$
FIFO n-n	$\top$	$\exists m = \langle \_, c_m, \_, \_, \_, h_{g_m} \rangle \in \texttt{NET} : c_m = c$ $\quad \wedge \neg \exists m' \in \texttt{NET} : m' \in h_{g_m}$
FIFO 1-n	$\top$	$\exists m = \langle \_, c_m, p_m, h_{l_m}, \_, \_ \rangle \in \texttt{NET} : c_m = c$ $\quad \wedge \neg \exists m' = \langle \_, \_, p_{m'}, \_, \_, \_ \rangle \in \texttt{NET} : p_m = p_{m'} \wedge m' \in h_{l_m}$
FIFO n-1	$\top$	$\exists m = \langle \_, c_m, \_, \_, \_, h_{g_m} \rangle \in \texttt{NET} : c_m = c$ $\quad \wedge \neg \exists m' = \langle \_, c_{m'}, \_, \_, \_, \_ \rangle \in \texttt{NET} : c_{m'} \in L \wedge m' \in h_{g_m}$
Causal	$\top$	$\exists m = \langle \_, c_m, \_, \_, h_{c_m}, \_ \rangle \in \texttt{NET} : c_m = c$ $\quad \wedge \neg \exists m' = \langle \_, c_{m'}, \_, \_, \_, \_ \rangle \in \texttt{NET} : c_{m'} \in L \wedge m' \in h_{c_m}$
FIFO 1-1	$\top$	$\exists m = \langle \_, c_m, p_m, h_{l_m}, \_, \_ \rangle \in \texttt{NET} : c_m = c$ $\quad \wedge \neg \exists m' = \langle \_, c_{m'}, p_{m'}, \_, \_, \_ \rangle \in \texttt{NET} : p_m = p_{m'} \wedge c_{m'} \in L \wedge m' \in h_{l_m}$
Async	$\top$	$\exists m = \langle \_, c_m, \_, \_, \_, \_ \rangle \in \texttt{NET} : c_m = c$

TABLE 4.2 – Gardes distinctives des différents modèles de communication pour les modèles TLA$^+$ avec histoires.

Modèle	send(p,d,m)	receive(p,m)
RSC	$\texttt{NET} = \varnothing$	$m \in \texttt{NET} \wedge destOf(m) = p$
FIFO n-n	$\top$	$m \in \texttt{NET} \wedge destOf(m) = p$ $\quad \wedge \neg \exists m' \in \texttt{NET} : m' \in hgOf(m)$
FIFO 1-n	$\top$	$m \in \texttt{NET} \wedge destOf(m) = p$ $\quad \wedge \neg \exists m' \in \texttt{NET} : origOf(m) = origOf(m') \wedge m' \in hcOf(m)$
FIFO n-1	$\top$	$m \in \texttt{NET} \wedge destOf(m) = p$ $\quad \wedge \neg \exists m' \in \texttt{NET} : destOf(m') = p \wedge m' \in hgOf(m)$
Causal	$\top$	$m \in \texttt{NET} \wedge destOf(m) = p$ $\quad \wedge \neg \exists m' \in \texttt{NET} : destOf(m') = p \wedge m' \in hcOf(m)$
FIFO 1-1	$\top$	$m \in \texttt{NET} \wedge destOf(m) = p$ $\quad \wedge \neg \exists m' \in \texttt{NET} : \wedge origOf(m) = origOf(m') \wedge destOf(m') = p \wedge m' \in hcOf(m)$
Async	$\top$	$m \in \texttt{NET} \wedge destOf(m) = p$

TABLE 4.3 – Gardes distinctives des différents modèles de communication pour les modèles Event-B avec histoires.

Ces gardes sont très proches. Seule diffère la gestion du destinataire du message. Dans le cas de réception de message sur un canal on vérifie que le canal de réception est bien le bon ($c_m = c$), dans le cas d'une réception avec destinataire explicite on vérifie que le destinataire est bien le bon ($destOf(m) = p$). Dans le cas d'une réception de message sur un canal on vérifie qu'il n'y a pas un autre message en transit qui intéresse le pair récepteur ($\neg \exists m' = \langle \_, c_{m'}, \_, \_, \_, \_ \rangle \in \texttt{NET} : c_{m'} \in L$), dans le cas d'une réception avec message explicite on vérifie qu'il n'y a pas un autre message qui est destiné au même pair ($\neg \exists m' \in \texttt{NET} : destOf(m') = p$). Lorsque les histoires locales sont utilisées dans les modèles TLA$^+$, ce sont les histoires causales qui sont utilisées dans les modèles Event-B. Ceci est possible, car nous avons aussi dans les gardes l'égalité des pairs d'émission des messages et ordre local et ordre causal se confondent pour un pair donné.

Un exemple de modèle en Event-B L'obtention des machines Event-B se fait en deux étapes de raffinement : d'abord superposer les histoires aux événements, puis supprimer les variables inutiles (celles liées aux événements). La figure 4.13 présente un extrait de la machine Event-B pour le modèle de communication FIFO 1-1 avec histoires.

Un exemple de modèle en TLA$^+$ Les modèles de communication avec histoires ont également été définis dans TLA$^+$ (et intégrés au framework de vérification). La figure 4.14 présente un extrait de la machine TLA$^+$ pour le modèle de communication FIFO 1-1 avec histoires.

4.4.1.3 Définitions basées sur des structures ad-hoc

Pour chacun des sept modèles de communication, des modèles plus concrets ont été définis. Ils sont basés sur des structures de données ad-hoc comme les compteurs, les files ou encore les vecteurs d'horloges. Ils ont été définis en Event-B et en TLA$^+$. Ils sont utiles d'une part pour le framework car il résolvent des soucis de performance et d'autre part pour montrer l'implémentabilité des modèles de communication. Leurs implantations ne sont pas détaillées ici, car l'accent est mis sur les représentations unifiées des modèles de communication qui permettent de les comparer.

4.4.2 Comparaison des différents modèles de communication

Les différents modèles présentés précédemment vont être comparés les uns aux autres. Cette comparaison est facilitée par les définitions uniformes de ceux-ci. La figure 4.15 résume les différents liens entre les modèles et les preuves réalisées.

4.4.2.1 Inclusion de modèles

Définition 4.4.1 (Inclusion de modèle)**.** *Un modèle de communication M_1 est inclus dans un modèle M_2 si et seulement si tous les calculs distribués conformes à M_1 sont aussi conforme à M_2 :*

$$M_1 \subseteq M_2 \triangleq \mathscr{C}alcul(M_1) \subseteq \mathscr{C}alcul(M_2)$$

Theorem 4.4.2 (Hiérarchie des modèles de communication)**.** *Le théorème est illustré par la figure 4.16 et formalisé comme suit :*

- *RSC $\subsetneq$ FIFO n-n $\subsetneq$ FIFO n-1 $\subsetneq$ Causal $\subsetneq$ FIFO 1-1 $\subsetneq$ Async*
- *RSC $\subsetneq$ FIFO n-n $\subsetneq$ FIFO 1-n $\subsetneq$ Causal $\subsetneq$ FIFO 1-1$\subsetneq$ Async*
- *FIFO 1-n et FIFO n-1 ne sont pas comparables.*

Démonstration. Voir [CHQ16].

■

MACHINE Fifo11History **REFINES** Fifo11History **SEES** Messages

VARIABLES

net // *Network*
hg // *Global history*
hc // *Causal history per peer*
origOf // *sender of message*
destOf // *destination of message*
hgOf // *global history of message*
hcOf // *causal history of message*

INVARIANTS

Tnet: net $\in \mathbb{P}(hg)$
Thg: hg $\in \mathbb{P}(MESSAGE)$
Thc: hc $\in PEER \rightarrow \mathbb{P}(hg)$
TorigOf: origOf $\in$ hg $\rightarrow$PEER
TdestOf: destOf $\in$ hg $\rightarrow$PEER
ThgOf: hgOf $\in$ hg $\rightarrow \mathbb{P}(hg)$
ThcOf: hcOf $\in$ hg $\rightarrow \mathbb{P}(hg)$
ordering : $\forall$ m1, m2 $\cdot$ m1 $\in$ hg $\wedge$ m2 $\in$ hg $\wedge$ m1 $\neq$ m2
$\wedge$ origOf(m1) = origOf(m2)
$\wedge$ destOf(m1) = destOf(m2) $\wedge$ m1 $\in$ hcOf(m2)
$\Rightarrow \neg$ (m1 $\in$ net $\wedge$ m2 $\notin$ net)

EVENT INITIALISATION THEN

a1: net := $\varnothing$
...
a9: hcOf := $\varnothing$

EVENT send **REFINES** send

ANY p m d
WHERE
Tp: p $\in$ PEER
Tm: m $\in$ MESSAGE $\setminus$ hg // *new message id*
Td: d $\in$ PEER
ordering : $\top$
THEN
a1: net := net $\cup$ {m}
a2: hg := hg $\cup$ {m}
a4: hc(p) := hc(p) $\cup$ {m}
a5: origOf := origOf $\cup$ {m$\mapsto$p}
a6: destOf := destOf $\cup$ {m$\mapsto$d}
a7: hgOf := hgOf $\cup$ {m$\mapsto$hg}
a8: hcOf := hcOf $\cup$ {m$\mapsto$hc(p)}

EVENT receive **REFINES** receive

ANY p m
WHERE
Tp: p $\in$ PEER
intransit : m $\in$ net
destination : destOf(m) = p
ordering : $\neg$ ($\exists$ m2 $\cdot$ m2 $\in$ net $\wedge$ origOf(m) = origOf(m2)
$\wedge$ destOf(m2) = p $\wedge$ m2 $\in$ hcOf(m))
THEN
a1: net := net $\setminus$ {m}
a2: hc(p) := hc(p) $\cup$ hcOf(m) $\cup$ {m}

FIGURE 4.13 – Machine Event-B pour le modèle de communication FIFO 1-1 avec histoires

MODULE *fifo*11

EXTENDS *Naturals*, *Defs*

$Init \triangleq id = 1 \land net = \{\} \land hl = [i \in Peer \mapsto \{\}] \land hc = [i \in Peer \mapsto \{\}] \land hg = \{\}$

$send(peer, chan) \triangleq$ The peer "peer" sends a new message on channel "*chan*"

$\land\ id' = id + 1$

$\land$ LET $m \triangleq \{id\} \times \{chan\} \times \{peer\} \times \{hl[peer]\} \times \{hc[peer]\} \times \{hg\}$ IN

$net' = net \cup m$

$\land\ hl' = [hl$ EXCEPT $![peer] = @ \cup \{id\}]$

$\land\ hc' = [hc$ EXCEPT $![peer] = @ \cup \{id\}]$

$\land\ hg' = hg \cup \{id\}$

$deliveryOk(m, listened) \triangleq \neg(\exists m2 \in net :$ **Ordering property.** There is no transiting message $m2$ from the same peer, whose channel is listened, and in the local history of m (thus previously sent by the same peer).

$\land\ mp(m) = mp(m2)$

$\land\ mc(m2) \in listened$

$\land\ mid(m2) \in mhl(m))$

$receive(peer, chan, listened) \triangleq$

$\exists m \in net :$ The peer "peer" receives a message on "*chan*", while listening to a set of channels "listened"

$\land\ chan = mc(m)$

$\land\ deliveryOk(m, listened)$

$\land\ net' = net \setminus \{m\}$

$\land$ UNCHANGED $\langle id, hl, hg \rangle$

$\land\ hc' = [hc$ EXCEPT $![peer] = @ \cup mhc(m) \cup \{mid(m)\}]$

$NextSend \triangleq \exists p \in Peer : \exists c \in Channel : send(p, c)$

$NextRecv \triangleq \exists p \in Peer : \exists c \in Channel : \exists l \in$ SUBSET $Channel : receive(p, c, l)$

$Next \triangleq NextSend \lor NextRecv$

$Spec \triangleq Init \land \Box[Next]_{vars}$

FIGURE 4.14 – Module TLA$^+$ pour le modèle de communication FIFO 1-1 avec histoires

Les travaux, de ce chapitre, se concentrent sur l'étude des calculs distribués. Cependant, une version de cette hiérarchie pour les exécutions distribuées a également été réalisée et exposée dans [SHQ17]. Les exécutions distribuées ne capturant pas l'ordre global, les liens entre les modèles plus contraints que le modèle causal ne sont plus les mêmes.

4.4.2.2 Un autre point de vue : le raffinement

Une des preuves présentées dans [CHQ16] (inclusion des modèles basés évènements) est une preuve manuelle basée sur l'implication des prédicats de délivrance. Une autre façon de réaliser la preuve d'inclusion est de montrer qu'un modèle raffine l'autre. En effet si un modèle M_1 raffine un modèle $M2$, M_1 ne générera pas plus d'exécution que M_2 et toutes les exécutions de M_1 seront compatibles avec M_2.

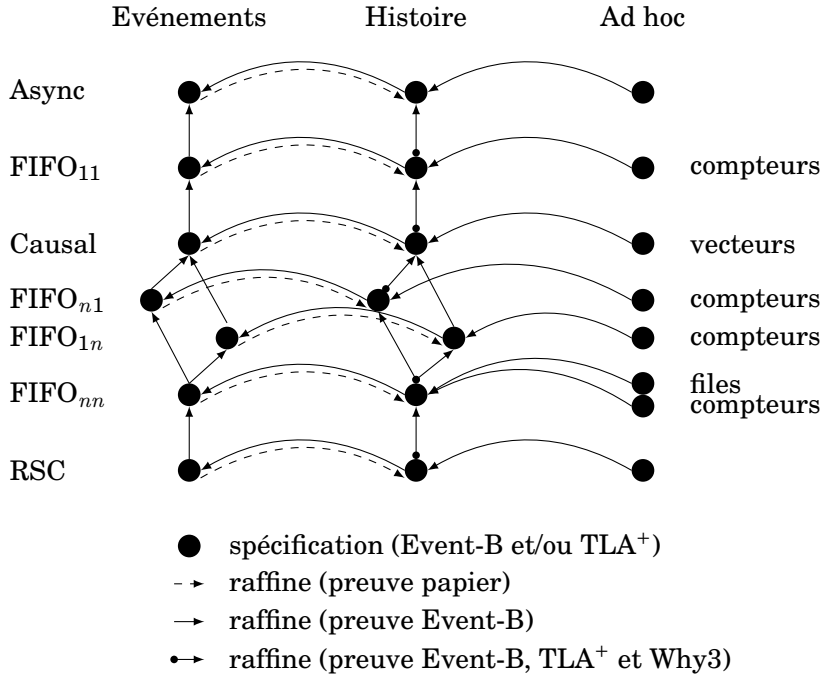


FIGURE 4.15 – Relations entre les modèles de communication

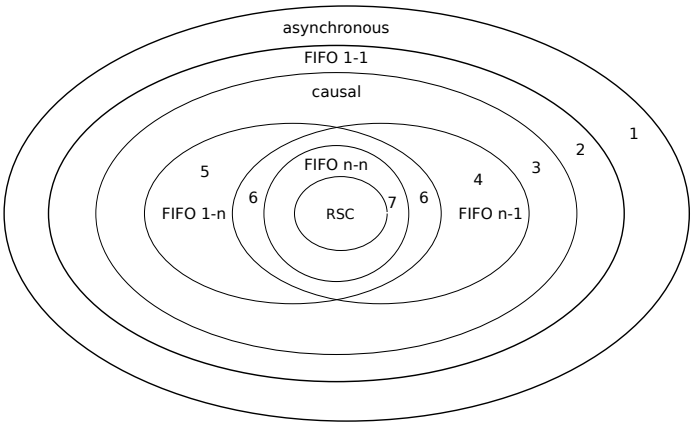


FIGURE 4.16 – Hiérarchie des modèles de communication

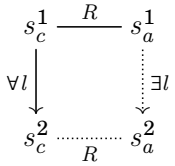
Il existe plusieurs notions de raffinements, nous allons en développer trois : la notion générale de simulation, le raffinement de TLA$^+$ et celui d'Event-B. Dans le cas de nos modèles, ces trois formes de raffinement ont été utilisées avec différents outils (Why3 pour la simulation, et les outils dédiés pour les deux autres) pour prouver les inclusions de modèles. Pour une même formalisation (basée événements ou histoires), la hiérarchie des modèles a été prouvée par raffinement :

- La hiérarchie des modèles Event-B basés événements a été prouvée avec Rodin et la notion de raffinement Event-B.
- La hiérarchie des modèles Event-B basés histoires a été prouvée avec Rodin et la notion de raffinement Event-B.
- La hiérarchie des modèles TLA$^+$ basés histoires a été prouvée avec Why3 et la notion de simulation.
- La hiérarchie des modèles TLA$^+$ basés histoires a été prouvée avec TLAPS et la notion de raffinement TLA$^+$.

Simulation et preuves Why3 Un modèle abstrait $M_a = (S_a, I_a, R_a, L)$ simule un modèle concret $M_c = (S_c, I_c, R_c, L) \triangleq \exists R \in S_c \times S_a$:

- $\forall i_c \in I_c, \exists i_a \in I_a, R(i_c, i_a)$
- $\forall s_c^1, S_c^2 \in S_c, s_a^1 \in S_a, l \in L, (R(S_c^1, S_a^1) \wedge R_c(s_c^1, l, S_c^2)) \Rightarrow (\exists s_a^2 \in S_a, R(S_c^2, S_a^2) \wedge R_a(s_a^1, l, S_a^2))$

Cette dernière propriété est illustrée par la figure suivante :



S'il existe une relation entre un état du système concret (s_c^1) et un état du système abstrait (s_a^1), alors pour toutes les transitions (l) depuis cet état du système concret, il doit exister une transition avec le même label depuis l'état du système abstrait et les états d'arrivée (s_c^2 et s_a^2) doivent être en relation.

Intuitivement, si un modèle abstrait M_a simule un modèle concret M_c, cela signifie que M_a fait au moins autant de chose que M_c. Toutes les exécutions de M_c sont également des exécutions de M_a. La réciproque n'est pas vraie.

Nous avons utilisé cette définition pour prouver la hiérarchie des modèles TLA$^+$ basés histoires avec Why3. Les preuves sont disponibles à cette adresse : http://hurault.perso.enseeiht.fr/asynchronousCommunication/refinements.why.

Les modèles TLA$^+$ basés histoires ont été ré-écrits en Why3 et les preuves de raffinement sont, à l'exception d'une, passées automatiquement grâce aux solveurs Alt-Ergo et CVC4. La

preuve posant problème est le raffinement entre $FIFO_{1-n}$ et $Causal$. Elle nécessite une propriété sur les messages en transit dans le modèle $FIFO_{1-n}$ (à savoir que deux messages en transit et causalement liés ont été émis par le même pair). Cette propriété n'a pas été prouvée avec Why3 mais est un corollaire du lemme 3.4 de [CHQ16].

Raffinement TLA$^+$ et preuves TLAPS En TLA$^+$ la spécification $Spec$ d'un module s'écrit : $Init \wedge \Box[Next] \wedge Fairness$ où $Init$ représente les états initiaux, $Next$ la fonction de transition et $Fairness$ les équités éventuelles nécessaires au modèle.

En TLA$^+$ un module concret M_c raffine un module abstrait M_a si la spécification du premier implique la spécification du second : $Spec_c \Rightarrow Spec_a$ soit $Init_c \wedge \Box[Next_c] \wedge Fairness_c \Rightarrow Init_a \wedge \Box[Next_a] \wedge Fairness_a$. Ceci n'a de sens que dans le cas où les deux modèles partagent les mêmes variables. Sinon, il faut définir une liaison entre les variables du module abstrait et celles du module concret.

Une particularité du raffinement TLA$^+$ est la conservation des propriétés de vivacité due à la présence de l'équité dans la formule à prouver.

Nos modèles TLA$^+$ basés histoires ayant les mêmes variables d'état, les mêmes actions et la même équité, la preuve de raffinement se réduit à prouver l'implication des prédicats d'initialisation et pour chaque action : $Act_c \Rightarrow Act_a$. Toutes les preuves TLAPS (disponibles ici http://queinnec.perso.enseeiht.fr/ABZ2016) sont de la même forme : d'abord le cas des états initiaux, puis le traitement de la fonction de transition. La preuve de l'implication des fonctions de transition étant elle-même décomposée en deux cas : envoi et réception de message.

Une preuve est manquante, celle du raffinement de $Causal$ par $FIFO_{1-n}$ (celle qui avait déjà posé problème à Why3). Elle n'est pas complète, il manque la preuve de deux invariants. En combinant ces deux invariants nous retrouvons la propriété qui avait dû être mise en hypothèse dans Why3.

Raffinement Event-B et preuves Rodin Le concept principal de la méthode Event-B est le raffinement. Soit une machine concrète M_c avec les variables w, une machine abstraite M_a avec les variables v, C_c et C_a les axiomes des deux machines et $I_c(w)$ et $I_a(v)$ les invariants des deux machines. Les variables v de la machine abstraite sont liées aux variables w de la machine concrète par l'invariant de liaison $J(v,w)$. Le raffinement d'un évènement E_a de M_a (avec les paramètres x, la garde $G_a(v,x)$ et l'action $BA_a(v,x,v')$) en un évènement E_c de M_c (avec les paramètres y, la garde $G_c(w,y)$ et l'action $BA_c(w,y,w')$) génère les obligations de preuve suivantes :

$$C_c \wedge C_a \wedge I_c(w) \wedge I_a(v) \wedge J(v,w) \wedge G_c(w,y) \wedge W(x,w,y) \Rightarrow G_a(v,x)$$
$$C_c \wedge C_a \wedge I_c(w) \wedge I_a(v) \wedge J(v,w) \wedge G_c(w,y) \wedge W(x,w,y) \wedge BA_c(w,y,w') \Rightarrow \exists v' \cdot (BA_a(v,x,v') \wedge J(v',w'))$$

$W(x,w,y)$ est le prédicat qui lie les paramètres de l'évènement abstrait et ceux de l'évène-

ment concret s'ils ne sont pas égaux. La première formule signifie que la garde est renforcée. La seconde formule signifie que l'action associée à l'évènement concret n'est pas contradictoire avec l'action associée à l'évènement abstrait. `skip` est l'évènement de bégaiement qui est implicitement raffiné par un évènement de la machine concrète qui ne raffine aucun autre évènement.

Le raffinement Even-B garantit qu'une transition concrète est légale dans l'abstrait et qu'il n'y a pas plus de blocage (deadlock) dans le modèle concret que dans le modèle abstrait.

La hiérarchie des modèles Event-B basés évènements a été prouvée avec Rodin. Les modèles sont exprimés de façon homogène, leur unique différence est l'invariant exprimant l'ordre de délivrance. Les preuves de raffinement consistent donc à prouver une implication logique entre ces invariants. Il n'a donc pas été nécessaire de définir d'invariant de liaison. La plupart du temps, ces preuves n'ont demandé que peu d'interventions manuelles grâce aux auto-provers, post-tactiques et solveurs SMT. Comme pour les preuves précédentes, le raffinement de $Causal$ par $FIFO_{1-n}$ a nécessité un invariant spécifique qui dit que la causalité entre deux évènements ne se produisant pas sur le même pair est forcément issue d'un échange de messages. Il a été prouvé à l'aide de Rodin.

La hiérarchie des modèles Event-B basés histoires a été prouvée avec Rodin. Cette fois encore, les modèles sont exprimés de façon homogène. D'un modèle à l'autre, seule la garde de l'évènement de réception diffère (sauf pour RSC). Les preuves de raffinement consistent donc à prouver l'implication des gardes. Là encore toutes les preuves sont passées avec très peu d'intervention manuelle et seul le raffinement de $Causal$ par $FIFO_{1-n}$ a été compliqué. Il nécessite, encore une fois, un invariant spécifique qui a été prouvé automatiquement avec Rodin.

Bilan Les différentes preuves ont été réalisées à plusieurs mains. Les preuves Why3 ont été les premières à être réalisées. Classiquement, la traduction des modèles en Why3 peut être questionnée, néanmoins ceux-ci étant relativement simples, la traduction ne laisse pas beaucoup de place à l'erreur. Les preuves de raffinement se ramenant à des preuves d'implication des conditions de délivrance, les solveurs SMT semblaient une bonne cible. Connaissant la facilité de Why3 à communiquer avec les SMT, Why3 a été utilisé pour avoir des preuves mécanisées "à moindre effort".

Les preuves TLAPS ont été les secondes à être réalisées, levant ainsi le problème de la traduction des modèles. Le travail réalisé pour les preuves Why3 (recherche de lemmes intermédiaires) a grandement facilité l'écriture de celles-ci. L'écriture des preuves TLAPS est un peu complexe, car elle nécessite pour chaque étape de preuve de préciser quels définitions ou lemmes utiliser. Néanmoins une fois terminées et validées, les preuves TLAPS sont lisibles par l'humain et très précises sur ce qui est nécessaire pour quelle partie de preuve. En opposition, avec Why3, nous savons uniquement que les solveurs ont validé les obligations de preuve, mais nous n'avons aucune idée de la structure de la preuve.

Les preuves des modèles Event-B avec Rodin ont été les dernières réalisées. Elles ont été relativement rapides, car une fois encore elles ont bénéficié des expériences précédentes. Même si les modèles ne sont pas identiques, ils ont de fortes similitudes, donc les lemmes / invariants nécessaires avaient déjà été identifiés. La même remarque est valide pour les preuves Rodin que pour les preuves Why3 : nous n'avons pas d'idée de leur structure.

Les preuves des modèles TLA$^+$ basés histoires ne sont pas complètes, alors que les preuves des modèles Event-B ont été finalisées. Plus qu'un problème d'outil (loin de moi l'idée de démontrer que Rodin est plus puissant / pratique / ... que Why3 ou TLAPS), la nature des modèles est probablement en jeu. Rappelons que les modèles TLA$^+$ n'ont pas de destinataire explicite, mais des canaux et une notion d'intérêt des pairs, alors que les modèles Event-B ont des destinataires explicites, ce qui rend cette notion d'intérêt inutile. Les modèles Event-B sont donc un peu plus simples à manipuler.

4.4.2.3 Correction des concrétisations d'un modèle

Pour un même modèle de communication, il est important de s'assurer que les versions basées sur des histoires et des structures de données ad-hoc sont correctes par rapport à la spécification abstraite à base d'événements et d'ordre.

Modèle avec destinataires explicites Pour les modèles avec destinataires explicites (Event-B) les preuves ont été réalisées par raffinement avec Rodin. Il s'agit ici d'un raffinement de concrétisation des structures de données : les événements (Event-B) ne sont plus modifiés (pas de nouveaux évènements créés, pas d'évènements supprimés et paramètres des évènements inchangés), seules les structures de données évoluent entre les modèles.

Les variables n'étant plus les mêmes il est nécessaire de définir des invariants de liaison pour lier les variables des modèles histoires avec celles des modèles évènements. Certains sont communs à tous les modèles. Par exemple l'invariant stipulant que *"un message qui est envoyé mais pas encore reçu (modèle basé évènement) est en transit et donc appartient au réseau (modèle basé histoire)"* est commun à tous les modèles.

Modèles sans destinataires explicites Pour les modèles avec envoi sur un canal (TLA$^+$) des preuves manuelles (preuves papier) [CHQ16] ont été réalisées pour la correction des modèles histoires vis-à-vis des modèles événements. Elles n'ont pas été mécanisées, car nous n'avons pas développé la version basée événements des modèles. Rien n'empêche leur définition. Historiquement les modèles TLA$^+$ sont liés au framework de vérification et se sont donc les versions opérationnelles (basées histoires ou avec structure de données ad hoc) qui ont été implantées.

Les envois de messages n'ayant pas de destinataires explicites, il a fallu ajouter une contrainte sur le système pour obtenir la correction. En effet, les pairs reçoivent les messages depuis les canaux pour lesquels ils ont déclaré leur intérêt. Il faut que cet intérêt soit décroissant.

Si un pair se met à être intéressé par un canal, rien ne garantit que l'ordre du modèle sera respecté. Par exemple, imaginons deux messages envoyés sur le canal a et sur le canal b depuis le même pair et dans cet ordre. Un autre pair qui ne serait dans un premier temps intéressé que par le canal b puis dans un second temps que par le canal a pourrait recevoir les messages dans cet ordre même dans le modèle $FIFO_{1-1}$, ce qui n'est pas correct. Le problème ne se pose pas quand les destinataires sont explicites, car ce n'est pas une information qui peut être modifiée contrairement à l'intérêt d'un pair qui est dynamique.

4.4.2.4 Complétude de la version histoire d'un modèle vis-à-vis de sa version événements

La correction des modèles n'est pas suffisante. Quand nous nous intéressons à la compatibilité d'un ensemble de pairs (notamment à l'aide du framework), modulo un modèle de communication, nous avons besoin d'être certain que ce modèle de communication est complet vis-à-vis de sa spécification, c'est-à-dire que tous les calculs distribués issus de ces pairs et compatibles avec les modèles de communication sont bien engendrés.

Une preuve de complétude des modèles histoires TLA$^+$ par rapport aux modèles événements a été réalisée [CHQ16]. La preuve de complétude nécessite que le système vérifie certaines propriétés : l'intérêt des pairs doit être décroissant et le système doit être mono-récepteur (c'est-à-dire qu'au plus un pair est intéressé par un canal) pour certains modèles (FIFO 1-1, causal et FIFO n-1). Le théorème garantit alors la génération de toutes les exécutions éventuellement $fairs$, c'est-à-dire toutes les exécutions qui peuvent être prolongées en une exécution où tout message envoyé est reçu.

Nous n'avons pas de résultat similaire pour les modèles Event-B. En effet pour cela, il faudrait faire le raffinement des modèles basés histoires par les modèles basés évènement. Pour que ces raffinements soient possibles, il faut modifier nos modèles basés évènements pour exhiber les contraintes nécessaires à la complétude. Les notions de décroissance de l'intérêt et de mono-récepteur n'ont pas de sens ici, puisque les modèles Event-B ont des destinataires explicites. Par contre les modèles basés évènements doivent être modifiés pour que seules les exécutions "prolongeables" soient prises en compte.

4.5 Modèles de communication applicatifs

Nous ne nous intéressons plus ici aux modèles génériques mais aux modèles applicatifs. L'idée est de donner une priorité relative aux canaux. Un message n'est pas délivrable s'il en existe un plus prioritaire sur le réseau.

4.5.1 Formalisation

Définition 4.5.1 (Réseau)**.** *A chaque étape d'un calcul distribué,* NET_c *est l'ensemble des canaux sur lesquels au moins un message est en transit.*

Définition 4.5.2 (Réception désactivée)**.** *La réception sur un canal* c *est désactivée (*$disabled(c?)$*) à un certain point d'un calcul si elle ne peut pas avoir lieu à ce point.*

Les priorités des canaux sont un ensemble de contraintes statiques qui forcent des réceptions à être désactivées en fonction de l'état du réseau. (B BLOCKS c) signifie que si au moins un message est en transit sur au moins un des canaux de B, alors la réception sur c est *désactivée*.

Définition 4.5.3 (BLOCKS)**.** *Un système* Sys *paramétré par l'ensemble de contraintes* BLOCKS C *respecte :*

$$Sys, C \models \forall B \subseteq Channels, \forall c \in Channels,$$
$$(B \text{ BLOCKS } c) \in C \implies (\forall b \in B, \Box((b \in NET_c) \implies disabled(c?)))$$

4.5.2 Framework

La vérification de système pour un modèle applicatif (ensemble de contraintes BLOCKS) est intégrée au framework TLA$^+$. Un message peut toujours être envoyé. Un message ne peut être reçu par un pair que s'il intéresse le pair et qu'il n'existe pas de message plus prioritaire, intéressant également le pair, sur le réseau.

Ces modèles applicatifs se combinent avec les modèles génériques.

4.5.3 Inférence

Il n'est pas intuitif de deviner l'ensemble de contraintes nécessaires pour garantir une propriété de compatibilité entre un ensemble de pairs. Un mécanisme d'inférence a donc été développé. Il fonctionne de façon itérative en appelant le framework et en analysant les contre-exemples en cas de non-compatibilité. Cette analyse de contre-exemple permet d'enrichir l'ensemble des contraintes.

4.5.3.1 Principe

L'algorithme d'inférence commence avec un ensemble de contraintes vide et va générer l'ensemble des ensembles de contraintes qui permettent de garantir qu'un système vérifie une propriété donnée. Le système ainsi que la propriété sont des paramètres de cet algorithme d'inférence.

Initialement l'ensemble de solutions candidates est vide. Le framework est alors utilisé pour étudier si le système vérifie la propriété. Dès que le framework génère un état du système où plusieurs réceptions sont possibles, il s'arrête et remonte un certain nombre d'informations à l'algorithme d'inférence. Ces informations sont analysées pour générer plusieurs solutions

candidates : certaines forcent une branche d'exécution en mettant une contrainte de priorité sur un message, d'autres laissent toutes les branches d'exécution accessibles. Ces solutions candidates permettent de générer toutes les exécutions initialement possibles.

Ce principe est alors itéré pour chaque solution candidate jusqu'à ce que les solutions soient soit validées (avec la solution candidate, le système valide la propriété), soient rejetées (avec la solution candidate, le système ne valide pas la propriété).

Cette version est pessimiste, puisqu'elle s'arrête à tous les points de choix, mais garantit d'obtenir toutes les solutions possibles. Une version optimiste existe, elle est plus rapide mais non exhaustive. Elle consiste à laisser le framework trouver une exécution qui viole la propriété. La trace de cette exécution est alors analysée et permet de générer des solutions candidates.

4.5.3.2 Exemple

Reprenons l'exemple du client-serveur de la section 4.3.2. Nous avons vu qu'une solution pour garantir qu'aucun message inattendu n'est délivré est d'imposer un modèle de communication au moins aussi strict que le modèle $Causal$. Une autre solution est d'imposer des contraintes applicatives.

L'algorithme d'inférence remonte les solutions suivantes (parmi d'autres) :

($\{start\}$ BLOCKS $upload$)	($\{start\}$ BLOCKS $accept$)	($\{start\}$ BLOCKS $accept$)
($\{upload\}$ BLOCKS end)	($\{upload\}$ BLOCKS end)	($\{upload\}$ BLOCKS $logout$)
($\{logout, end\}$ BLOCKS $login$)	($\{logout\}$ BLOCKS $login$)	($\{logout, end\}$ BLOCKS $login$)
États différents : 298	($\{end\}$ BLOCKS $start$)	**États différents :** 99
	États différents : 465	

Il faut noter que ces solutions ne sont pas comparables : aucun ensemble de contraintes n'est inclus dans un autre. Cela rend difficile le choix de l'ensemble de contraintes à appliquer. De plus ces solutions présentent une grande variation dans le nombre d'états du système, signifiant que certaines solutions autorisent plus d'exécutions que d'autres. Ce critère n'est pas non plus pertinent pour choisir l'ensemble de contraintes à appliquer. En effet, même si une solution présente un nombre d'états plus important, ce ne sont peut-être pas les états "intéressants" du système qui sont conservés.

4.6 Bilan

Dans ce chapitre la diversité des modèles de communication point à point asynchrone a été étudiée. Il présente un cadre unifié pour décrire les différents modèles et rendre explicite le type de communication utilisé au sein d'une composition. Ces modèles peuvent être génériques (les ordres de réception dépendent uniquement des ordres d'émission et des pairs d'émission ou réception) ou applicatifs (pour une application donnée, des priorités sont données entre les canaux d'envoi des messages). Un framework de vérification de composition de services (pairs)

est défini en TLA$^+$ et permet de vérifier (vérificateur de modèle) que la composition des pairs et d'un modèle de communication valide une propriété exprimée en LTL. La propriété vérifiée ainsi que le modèle de communication sont des paramètres de l'algorithme de vérification. Certains modèles de communication sont prédéfinis mais l'utilisateur peut définir son propre modèle tant qu'il respecte l'interface définie. Ces modèles ont été comparés entre eux et une hiérarchie établie.

4.6.1 Outils développés

Description basée sur les systèmes de transition La description des pairs et des modèles de communication est, classiquement, basée sur des systèmes de transition. Cette description formelle à l'avantage d'être supportée par de nombreux outils, qui permettent à la fois de réaliser des preuves mécanisées des résultats sur lesquels s'appuient les travaux (hiérarchies des modèles et correction des modèles basés histoires vis-à-vis des modèles basés évènements) et de fournir une mécanisation de l'algorithme de vérification de compatibilité (le framework en TLA$^+$). Cette description à base de systèmes de transitions permet également de définir la notion de composition des pairs grâce à la notion de produit synchronisé entre l'ensemble des pairs et un modèle de communication. Cette définition lève toute ambiguïté sur les propriétés de la communication entre les pairs puisque celle-ci est isolée et donc spécifiée indépendamment du comportement des pairs.

Framework de vérification Le framework, écrit en TLA$^+$, permet de vérifier automatiquement qu'une composition de services vérifie une propriété. Le framework étant basé sur un vérificateur de modèle, cette vérification n'est bien sûr possible que lorsque le nombre d'états du système est fini. Néanmoins, il est possible dans le cas infini, d'utiliser l'assistant de preuve TLAPS pour réaliser des preuves du système.

Inférence des modèles de communication Dans le cas des modèles applicatifs, un algorithme d'inférence de l'ensemble des modèles de communication, pour lesquels une composition vérifie une propriété, a été développé. Il révèle que les modèles trouvés ne sont pas forcément triviaux et surtout non comparables entre eux.

4.6.2 Diversité de la communication asynchrone

La communication asynchrone peut se décliner sous différentes formes. Dans certains travaux, il n'est pas toujours clair de savoir quel modèle de communication asynchrone est utilisé ni quelles sont les hypothèses faites sur celui-ci. Un des résultats importants est la cartographie des modèles de communication asynchrone. Elle permet de sensibiliser à leur variété, d'énoncer les propriétés et de les hiérarchiser (en terme de réduction de l'indéterminisme qu'ils permettent). Dans ces travaux, l'accent a été mis sur la possibilité de décrire cette diversité du

Émission / Réception	$\prec_p$	$\prec_\sigma$
$\prec_p$	$FIFO_{11}$	$FIFO_{1n}$
$\prec_c$	$Causal$	Prouvé égal à $FIFO_{1n}$
$\prec_\sigma$	$FIFO_{n1}$	$FIFO_{nn}$

FIGURE 4.17 – Modèles génériques asynchrones point à point

monde asynchrone. Différentes politiques d'ordonnancement des messages ont été étudiées, avec la possibilité de les décliner avec plusieurs multiplicités (point à point, diffusion et convergence) et avec des descriptions de différents niveaux d'abstraction. Même si toutes ces possibilités n'ont pas été développées, le cadre formel défini a été réfléchi pour pouvoir être étendu dans toutes ces directions.

Diversité des modèles génériques asynchrones point à point Usuellement, trois ordres sont utilisés dans les systèmes distribués : l'ordre local ($\prec_p$) à un pair, l'ordre causal ($\prec_c$) et l'ordre global ($\prec_\sigma$) à un calcul. Ces trois ordres peuvent engendrer différentes propriétés sur la communication : si deux évènements d'émission sont ordonnés selon un ordre, alors les évènements de réception associés sont ordonnés selon un second ordre. Les modèles imposant des réceptions causalement ordonnées n'ont pas été étudiés car jugés non réalistes d'un point de vue opérationnel. La combinaison des différents ordres en émission et réception permet donc d'engendrer six propriétés (modèles) de communication différent(e)s. Parmi ces six modèles, deux ont été prouvés égaux. Les différents modèles et leurs noms utilisés dans ce manuscrit sont exposés dans la figure 4.17. Ils ont été complétés par le modèle purement asynchrone qui n'impose aucune contrainte d'émission et de réception, et le modèle RSC qui se rapproche du modèle synchrone.

En plus de ces modèles, des modèles exploitant des priorités sur les canaux ont été étudiés.

Exhaustivité des multiplicités : point à point, diffusion, convergence Même si ce manuscrit se concentre sur la communication point à point, le cadre formel a été pensé pour prendre en compte les différentes multiplicités : le point à point (une émission est liée à une réception), la diffusion (un même message peut être reçu par plusieurs pairs) et la convergence (une même réception peut être liée à des émissions sur plusieurs pairs). Les résultats sur le point à point ont d'ailleurs été généralisés à la diffusion [Che17]. La convergence étant moins couramment utilisée, elle n'a pas été étudiée, mais les travaux s'étendraient sans doute sans difficulté à celle-ci.

Graduation des descriptions entre axiomatisation et opérationnalité Plusieurs niveaux d'abstraction ont été proposés pour la description de chaque modèle de communication. Cela permet d'avoir un large choix de modèles à disposition. Certains sont plus adaptés à la

preuve (modèles abstraits), d'autres sont plus proches de l'implantation (modèles utilisant des structures de données ad hoc telles que les files, compteurs. . .).

4.6.3 Comparaison des modèles de communications

Lorsque que l'on aborde le monde asynchrone dans sa diversité en définissant un grand nombre de modèles de communication, comme nous l'avons fait, il est naturel de vouloir les comparer entre eux.

Uniformité des descriptions Ces comparaisons ont été réalisées dans deux dimensions. Les différentes représentations d'un même modèle ont été vérifiées "cohérentes" grâce à la notion de correction prouvée par le raffinement et la notion de complétude. Les différents modèles de communication ont également été comparés entre eux. Une description uniforme de ceux-ci est nécessaire pour faciliter cette comparaison. Les différents modèles ont été décrits de façon uniforme à deux niveaux d'abstraction différents permettant de prouver une hiérarchie entre eux.

Mécanisation des preuves La mécanisation est une partie importante des travaux présentés. Elle a deux aspects : le fait de fournir des outils, mais également le fait de réaliser des preuves mécanisées. Le choix des descriptions et des formalismes a été en partie guidé par les outils qui les supportaient. Ayant dans l'équipe des compétences en TLA$^+$ et Event-B, c'est assez naturellement que les travaux ont été modélisés dans ces deux formalismes et que les preuves ont été mécanisées à l'aide de Rodin et/ou de TLC et TLAPS. Certaines preuves ont également été réalisées en utilisant Why3.

4.6.4 Expressivité

Isolation de la communication Les preuves de compatibilité / correction des algorithmes distribués (qui sont des compositions de pairs / services) sont souvent réalisées pour un modèle de communication donné et sont à refaire intégralement pour un autre modèle car l'algorithme et la communication sont étroitement liés. En voyant le modèle de communication comme une boîte noire dont les propriétés sont connues et en l'isolant du reste de l'algorithme, il est alors plus simple de définir les propriétés du modèle de communication nécessaires à la preuve et donc de connaître l'ensemble des modèles de communication pour lesquels une composition vérifie une propriété.

Canaux versus destinataires explicites Lors de la description des pairs, les canaux apparaissent naturellement, notamment car ils sont intégrés à des calculs de processus très utilisés comme CCS. Pourtant, leur présence n'est ni anodine ni évidente. Dans une partie des travaux, ils ne sont pas présents et sont remplacés par des destinataires explicites. En effet, lorsque l'on

s'intéresse uniquement à la communication et à la comparaison des modèles de communication entre eux, les canaux n'apportent rien et complexifient l'écriture des modèles et donc la réalisation des comparaisons. Par contre, lorsqu'il s'agit de décrire un système composé de pairs, il est naturel d'utiliser des canaux. Cela permet d'abstraire le destinataire du message et d'avoir une plus grande expressivité. En effet, des exemples d'utilisation aussi standard qu'une application client / serveur nécessitent de ne pas connaître explicitement le destinataire (le client s'adresse à un service sans avoir à s'occuper d'une éventuelle réplication des serveurs réalisant le service). Cela permet également d'utiliser la notation CCS pour décrire les pairs, ce qui est plus agréable que d'avoir à écrire le système de transition du pair. Pour toutes ces raisons, nous n'avons pas tranché entre les canaux et les destinataires explicites. Dans ces travaux les deux coexistent. Les canaux sont présents dans les modèles qui servent au framework de vérification de compatibilité de pairs, car ils permettent une description des pairs naturelle. Lorsqu'il s'agit de se concentrer sur la communication (indépendamment des pairs) et de comparer les différents modèles de communication, les canaux ne sont pas présents et les messages sont envoyés à des destinataires explicites.

Exécutions distribuées versus calculs distribuées Finalement, la dernière problématique à laquelle nous avons été confrontés est la dualité entre les exécutions distribuées (basées sur la connaissance de l'ordre causal des évènements) et les calculs distribués (basés sur la connaissance de l'ordre global des évènements). Si dans les systèmes distribués, sans mécanisme de centralisation, les pairs ne peuvent pas avoir plus de connaissance que la causalité, notre volonté d'exhaustivité des modèles génériques point à point nous a amené à considérer des modèles basés sur l'ordre global. Dans ces travaux nous nous intéressons donc exclusivement aux calculs distribués, mais des extensions de ceux-ci se sont concentrées sur les exécutions destribuées [SHQ17].

CHAPITRE 5

SÉLECTION DES COMPOSITIONS DE SERVICES BASÉE SUR LA QUALITÉ DE SERVICE

Ce chapitre s'articule autour de la thématique de la sélection des compositions de services en utilisant des critères de qualité de service. Lorsqu'un ensemble de compositions de services répondant au besoin fonctionnel est connu, il faut choisir celle qui sera exécutée. Le but ici est de choisir celle qui a les meilleures propriétés pour la plateforme d'exécution visée. Les propriétés ciblées dans ce chapitre sont les propriétés de qualité de service.

Dans ce chapitre, la qualité de service des services est décrite à l'aide de valeurs numériques représentant le temps d'exécution, le coût, la réputation, la disponibilité ou la fiabilité. Les premiers travaux présentés ne s'intéressent qu'au temps d'exécution, alors que les seconds s'intéressent aux cinq critères (agrégés ou non). Ces travaux s'étendraient à un nombre quelconque de critères de qualité de service.

Cette thématique est moins approfondie que les deux précédentes, elle vient en complément, presque en "bonus". Le but initial était de pouvoir choisir la meilleure composition de services parmi celles renvoyées par le courtier. Les compositions étant toutes correctes vis-à-vis du besoin fonctionnel, ce sont les aspects non fonctionnels qui déterminent le critère de qualité des compositions. La première partie de ce chapitre étudie la sélection de la meilleure (en terme de temps d'exécution) composition de services. Le temps d'exécution d'une composition est estimé à l'aide d'algorithmes d'apprentissage. La seconde partie du chapitre présente une approche qui combine la découverte et la sélection de la composition de services à l'aide d'un (ou plusieurs) algorithme(s) d'optimisation. Cela évite de générer l'ensemble des compositions répondant au besoin pour ensuite en sélectionner une unique à exécuter.

5.1 Sélection *a posteriori* à l'aide de l'apprentissage automatique

Travaux en collaboration avec l'université d'Hawaï et en particulier avec Henri Casanova. Ils ont donné lieu à la publication suivante : [HBC15].

Dans ce travail nous répondons à la question : étant donné un calcul qui peut être réalisé via des appels à des services standards d'algèbre linéaire, est-il possible de déterminer automatiquement la composition de services qui le réalise avec la meilleure performance ? Nous définissons la performance comme *le temps d'exécution*, c'est-à-dire le temps nécessaire pour réaliser le calcul. Nous étudions cette question en supposant des exécutions séquentielles monoprocesseur sur un ordinateur donné. Notre approche consiste en deux phases :

- une phase d'énumération utilisant le courtier (voir section 3.1) en boîte noire ;
- une phase de sélection de la composition de services la plus rapide en se basant sur une prédiction du temps d'exécution.

Une approche possible pour la sélection est de développer des modèles analytiques des coûts des services et de leurs compositions. Cependant, au vu de la complexité des logiciels modernes et du hardware, les modèles de performance sont approximatifs au mieux, même pour les exécutions mono-processeur. A la place, nous prédisons le temps de réponse en utilisant des techniques d'apprentissage automatique supervisé basé sur un ensemble de benchmarks d'exécutions de services et de composition de services. Notre but dans ce travail n'est pas de faire une avancée dans le domaine de l'apprentissage automatique, mais plutôt d'établir si les algorithmes d'apprentissage automatique peuvent être utilisés pour résoudre le problème de la sélection de la composition de services.

5.1.1 Méthodologie

Pour réaliser notre "proof of concept", nous avons choisi un sous-ensemble restreint de sept services disponibles issus des librairies BLAS [BDD+02] et LAPACK [ABB+99]. Même si cela peut sembler limité, le travail réalisé est facilement généralisable à un plus grand nombre de services.

5.1.1.1 Construction de l'espace de caractéristiques (feature space)

La première étape est de définir une représentation des solutions qui est utilisée pour entraîner les algorithmes d'apprentissage. Le bon apprentissage des algorithmes est conditionné par cette représentation. L'espace de caractéristiques doit capturer les caractéristiques principales d'une composition qui influent sur le temps d'exécution. Plus précisément nous associons une solution (un ensemble d'appel de services avec leurs paramètres comprenant entre autre

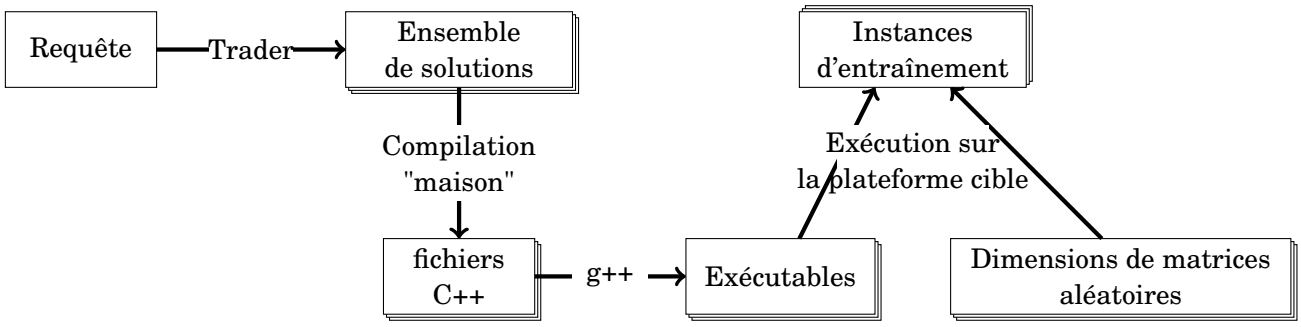


FIGURE 5.1 – Génération des données d'entraînement issues d'une requête au courtier.

les tailles des matrices en entrée et en sortie) avec un vecteur. Un ensemble de tels vecteurs associés à des mesures de temps d'exécution est utilisé pour entraîner les algorithmes d'apprentissage automatique qui produisent ensuite une prédiction de temps d'exécution pour un vecteur arbitraire.

Notre construction de cet espace de caractéristiques est guidée par l'exemple classique de la multiplication de trois matrices : $A \times B \times C$. A moins que les matrices ne soient carrées, une des deux solutions possibles, $(A \times B) \times C$ ou $A \times (B \times C)$, conduit à un temps de réponse plus court que l'autre. Par exemple, pour trois matrices A, B et C de taille respective $1 \times 10,000$, $10,000 \times 1$, et $1 \times 10,000$, le temps de calcul de $A \times (B \times C)$ est plus de 7000 fois plus important que celui de $(A \times B) \times C$ (dans nos conditions d'expérimentation). Cela est dut au fait que la matrice $B \times C$ est de taille $10,000 \times 10,000$ alors que la matrice $A \times B$ est de taille 1×1.

La définition de l'espace de caractéristiques n'étant pas triviale, deux espaces de caractéristiques ont été définis afin de pouvoir comparer leurs performances. Le premier espace est un vecteur de $n+3$ composants, où n est le nombre de services disponibles. Les n premiers composants sont, pour chacun des services, le nombre d'appels de celui-ci. Le vecteur est complété avec les tailles moyennes, minimales et maximales des dimensions des matrices. Le second espace est un vecteur de caractéristiques de $5n$ composants. Pour chacun des n services, il y a 5 composants : le nombre d'appels au service, les tailles moyennes, minimales et maximales des dimensions des matrices pour les appels à ce service et une mesure approximative de la complexité du service (basée sur sa complexité théorique).

5.1.1.2 Construction des données d'entraînement

Les données d'entraînement forment un ensemble d'instances de l'espace de caractéristiques qui permet aux algorithmes d'apprentissage automatique de construire leur modèle de coût. Il doit donc être représentatif des requêtes ultérieures.

Chaque donnée d'entraînement correspond à une composition de services instanciée avec des tailles de matrices, augmentée du temps de réponse mesuré sur notre ordinateur de test. Pour

une composition, nous mesurons le temps de réponse pour 50 instances de tailles de matrices aléatoires, échantillonnées uniformément entre 1 et 500.

Notre ensemble de données d'entraînement inclut des instances qui correspondent à un appel d'un service simple, pour un total de $7 \times 50 = 350$ instances. Nous avons également complété les données d'entraînement en générant des requêtes pour le courtier et en injectant les réponses du courtier selon le schéma 5.1. Ces requêtes ont été obtenues en combinant deux opérations parmi celles largement utilisées en algèbre linaires à savoir : l'addition ($+$), la multiplication ($\times$), l'inverse ($^{-1}$), la transposition (^{t}), et la factorisation LU (lu). Ces requêtes ont été déclinées avec des propriétés de matrice particulières : générale, symétrique et triangulaire.

Au total, les données d'entraînement comportent 22,750 instances (dont 350 instances issues d'une invocation d'un service seul).

5.1.1.3 Construction des données de test

Nous avons étudié différents domaines d'application de l'algèbre linéaire et avons choisi 7 requêtes issues du traitement d'image.

Ces requêtes ont également été traitées selon le schéma présenté dans la figure 5.1, mais en étant instanciées avec des tailles de matrices entre 1 et 10,000 (contre 500 pour les données d'entraînement).

5.1.2 Résultats

Pour plus de généralité dans les résultats, les expériences ont été réalisées deux fois, avec deux librairies d'algèbre linéaire différentes : ATLAS [WPD01] et OpenBLAS [1]. Nous avons obtenu des résultats similaires, ce qui permet de suggérer que l'approche n'est pas biaisée par l'utilisation d'une librairie particulière. Les chiffres et courbes montrées dans la suite sont ceux obtenus avec OpenBLAS.

5.1.2.1 Les algorithmes avec de bonnes prédictions

Nous nous sommes intéressés à savoir si les algorithmes d'apprentissage automatique trouvent souvent la meilleure composition de services et dans la mesure où ils se trompent, s'ils se trompent de beaucoup. Les résultats sont montrés sur les courbes des figures 5.2 et 5.3. Pour chaque algorithme 20 points sont montrés, chacun avec un ensemble de données d'entraînement différentes (tailles de matrices différentes).

Les croix vertes (x) représentent les données pour l'espace de caractéristiques complexe, tandis que les plus rouges (+) représentent les données pour l'espace de caractéristiques simple. La figure 5.2 représente le pourcentage de fois où l'algorithme choisit la meilleure solution. La ligne noire représente le pourcentage de fois où la première solution renvoyée par le courtier est

1. http://www.openblas.net/

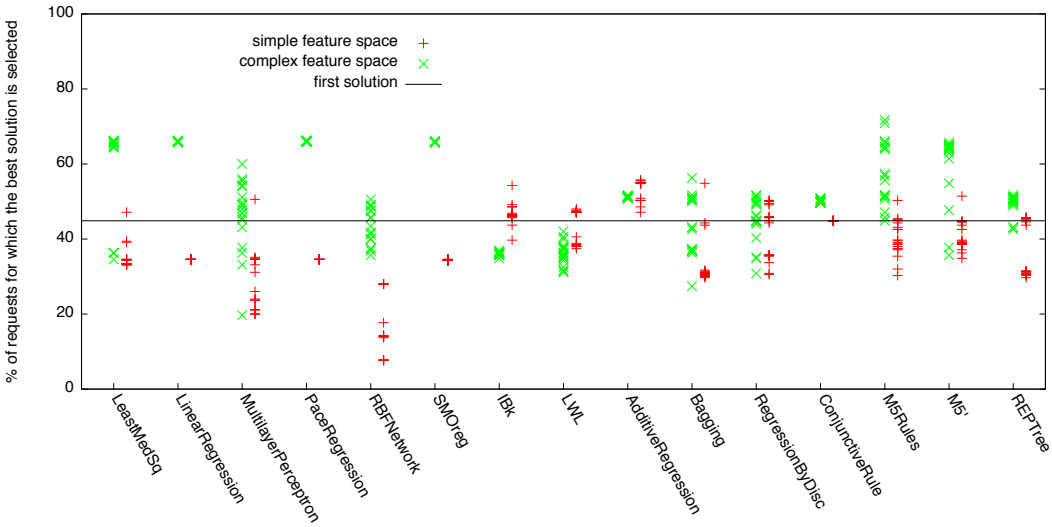


FIGURE 5.2 – Est-ce que les algorithmes choisissent la meilleure solution ?

la meilleure et permet de se comparer à un algorithme trivial et rapide qui consisterait à choisir d'exécuter la première solution du courtier. Quand un algorithme ne choisit pas la meilleure solution, il est important de savoir si la solution choisie est proche de la solution optimale ou pas. C'est ce que représente la figure 5.3. Elle représente l'erreur relative entre la solution choisie par l'algorithme et la solution optimale (en excluant les cas où l'algorithme trouve la meilleure solution). Plus l'erreur est proche de 1, meilleure est la solution.

Les courbes montrent nettement que l'espace de caractéristiques le plus complexe (c'est-à-dire celui avec $5n$ composantes) est celui qui donne les meilleurs résultats. Dans la suite nous nous concentrerons donc sur les résultats de celui-ci.

Parmi les divers algorithmes testés, l'algorithme de régression linéaire est celui qui s'est le mieux comporté (voir figure 5.2). Il est très régulier dans ses réponses et dans 65% des cas il trouve la meilleure solution. De plus quand il se trompe, il se trompe peu (voir la figure 5.3).

Deux autres algorithmes ont de bons résultats : PaceRegression et SMOReg. Ils sont, comme l'algorithme de régression linéaire, des algorithmes qui construisent des fonctions de coût (en opposition par exemple à ceux qui construisent des arbres de décision) à l'aide de méthodes de régression.

5.1.2.2 Comparaison avec Octave

Comme il a été vu dans la section 3.2.1, une proposition intéressante est d'intégrer le courtier avec des environnement interactifs de calculs d'algèbre linéaire tels que Matlab [MAT10], Scilab [Sci12], ou GNU Octave [Han11]. Des expériences simples sur ces trois outils ont permis de révéler qu'aucun d'eux ne prend en compte le cas pathologique de la multiplication expliqué

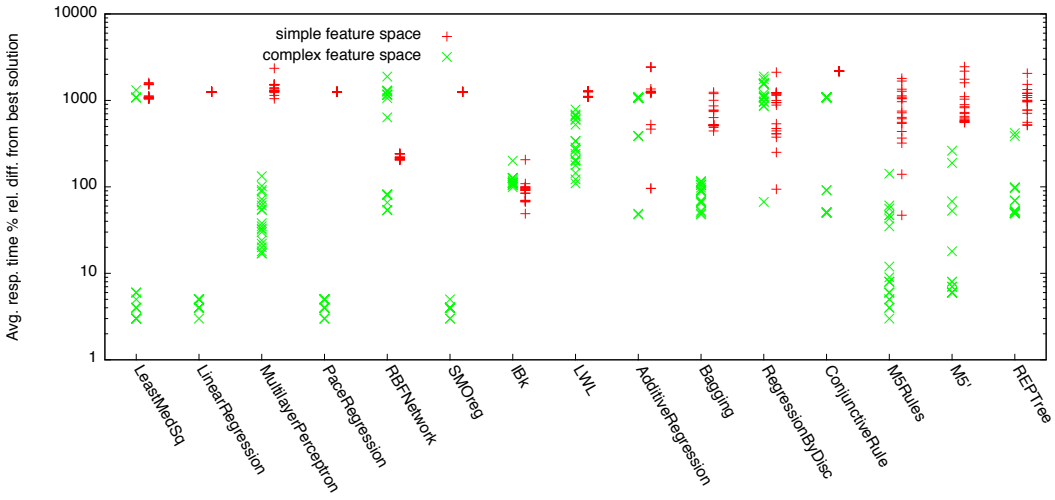


FIGURE 5.3 – De combien les algorithmes se trompent-ils ?

précédemment, laissant à l'utilisateur la responsabilité de bien présenter ses calculs. Dans cette partie nous étudions l'impact d'une potentielle intégration du courtier dans Octave.

Nous nous intéressons donc à l'utilisation du courtier combiné avec la sélection d'une composition parmi celles proposées. La sélection est réalisée avec l'aide de l'algorithme d'apprentissage automatique qui s'est le mieux comporté pour notre problème, à savoir celui de régression linéaire. Pour ce faire nous nous sommes comparés avec Octave. La figure 5.4 représente la différence relative entre le temps d'exécution d'Octave et le temps d'exécution du courtier et de la solution choisie (le temps de choix de sélection n'est pas compté car négligeable puisque nous considérons la phase d'entraînement des algorithmes d'apprentissage automatique comme hors ligne). Pour chaque requête, 50 points sont montrés, triés par temps croissant d'exécution d'Octave. Ils représentent 50 fois le même calcul mais avec des matrices différentes aussi bien en taille qu'en contenu. Les données au dessus de 250% ne sont pas montrés pour que le graphe reste lisible. Lorsque la différence relative est positive, cela signifie qu'Octave est plus rapide que le courtier combiné à la sélection. Inversement, lorsque la différence relative est négative, cela signifie qu'Octave est plus lent que le courtier combiné à la sélection.

Si nous nous intéressons uniquement au choix de l'algorithme d'apprentissage automatique (courbe rose et continue), nous nous rendons compte que le choix de l'algorithme d'apprentissage est souvent pertinent puisqu'il permet une réalisation du calcul plus rapide qu'Octave. Par contre, le temps d'énumération du courtier étant très lent, le bénéfice de l'approche est perdu (courbe orange et en pointillée où le temps d'énumération a pourtant été divisé par 20).

Il faut tout de même noter que le courtier a été réalisé comme un "proof of concept" et n'a jamais été optimisé.

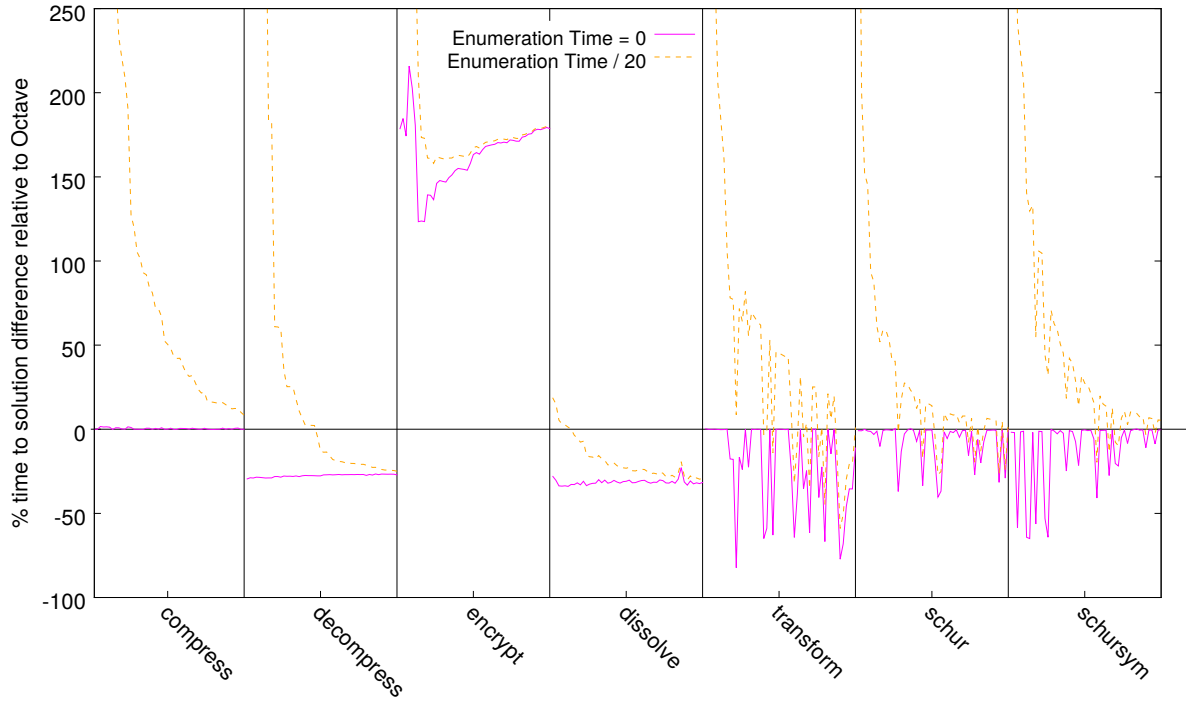


FIGURE 5.4 – Comparaison avec Octave.

5.2 Construction guidée par la qualité de service multi-critères

Collaboration avec l'université de Constantine II. Ces travaux sont en partie les travaux de thèse de Sériel Boussalia (Mai 2014 - Mai 2016), que j'ai co-encadrée avec Allaoua Chaoui. Ils ont donné lieu aux publications suivantes : [BCH+16, BCH15]

Ces travaux s'intéressent à la construction des compositions de services guidée par la qualité de service. Cela permet d'éviter l'écueil des travaux précédents : une phase chronophage de recherche exhaustive des compositions. L'idée est ici d'aller directement "à l'essentiel" en guidant, à l'aide d'algorithmes d'optimisation, la recherche de la composition dans la direction de la "meilleure" composition.

Peu de temps (d'espace) est consacré à ces travaux, car si les idées initiales sont intéressantes, la mise en œuvre n'est pas aboutie.

5.2.1 Description de la qualité de service des services

Comme pour tous les travaux précédents, avant de parler de composition de services validant une propriété, il faut pouvoir décrire cette propriété et donc proposer une description des services, de la composition de services et de la propriété.

5.2.1.1 Critères de qualité de service

Classiquement, la qualité de service est décrite par plusieurs valeurs :

- le temps d'exécution ;
- le coût (pécuniaire) ;
- la réputation (système de notation sur internet par exemple) ;
- la disponibilité et
- la fiabilité.

La qualité de service de la composition est décrite par les mêmes valeurs, calculées à partir des valeurs de qualité de service des différents services qui constituent la composition. Le temps d'exécution (resp. le coût) est la somme des temps d'exécution (resp. des coûts). La réputation est la moyenne des réputations. La disponibilité (resp. la fiabilité) est le produit des disponibilités (resp. des fiabilités).

5.2.1.2 Mono-critère vs. multi-critères

Pour optimiser cette qualité de service (tuple de valeurs), il y a deux approches possibles :

- agréger ces différentes valeurs (avec des pondérations pour laisser à l'utilisateur le choix de ses priorités) et optimiser un unique critère ;
- utiliser des méthodes d'optimisation multi-critères et proposer plusieurs solutions appartenant au front de Pareto. Les solutions appartenant au front de Pareto sont celles pour lesquelles il n'existe pas de solution améliorant un critère sans en dégrader un autre. A l'utilisateur ensuite de choisir la solution qui lui convient le mieux.

La première approche est plus simple mais impose d'agréger ensemble des critères qui ne sont pas comparables. Certes la pondération donne un degré de liberté important mais il reste difficile de définir un critère de qualité de service qui a du sens. La seconde approche, en n'agrégeant pas les différents paramètres de qualité de service, évite l'écueil de la solution précédente. Cependant les algorithmes d'optimisation multi-critères sont plus complexes et le choix final est laissé à l'utilisateur, ce qui est problématique dans une optique d'automatisation.

5.2.2 Services avec une entrée et une sortie

Dans un premier temps, nous nous sommes intéressés à un cas particulier permettant de faire une découverte à la volée du workflow d'exécution et de choisir la composition en optimisant un ou plusieurs critères de qualité de service.

Les services considérés possèdent une unique entrée et une unique sortie caractérisées par leur type. La requête de l'utilisateur fournit le type d'entrée, le type de sortie ainsi que la qualité de service souhaitée. L'algorithme construit une chaîne de services répondant à la problématique en utilisant des algorithmes d'optimisation pour guider la recherche de la composition optimale.

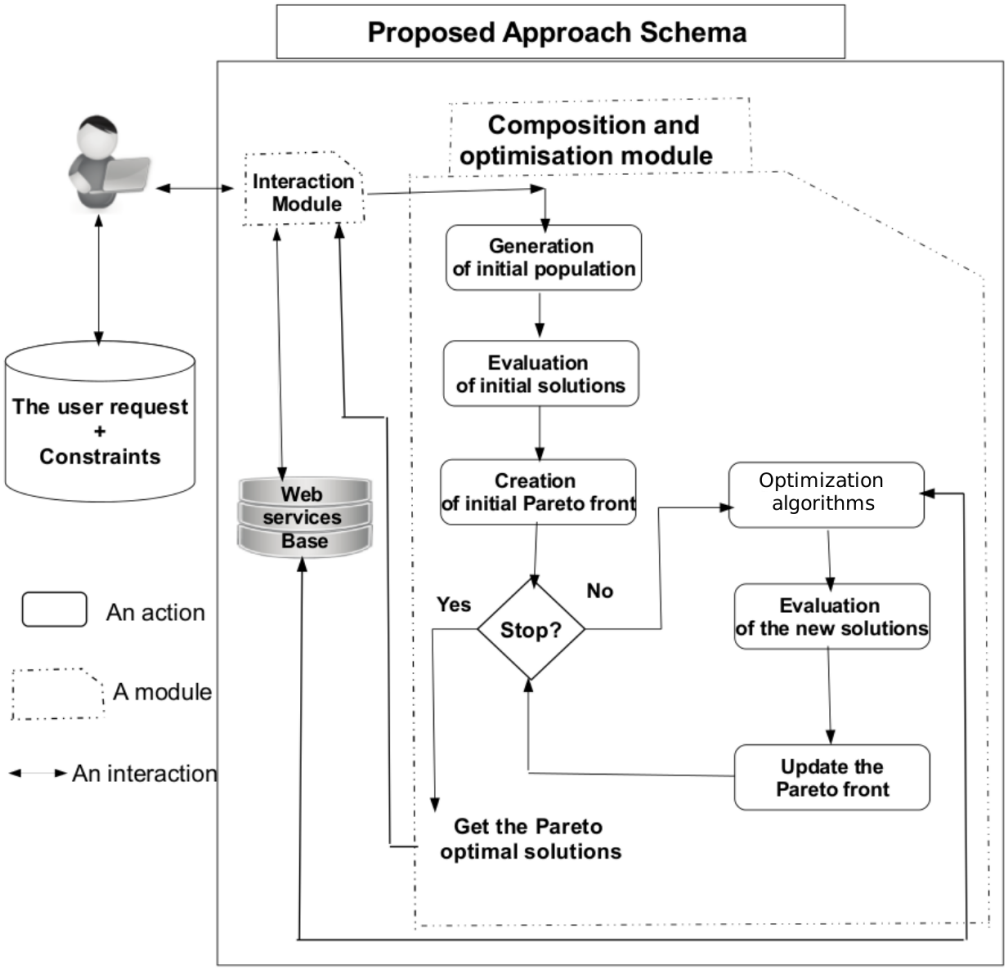


FIGURE 5.5 – Schéma de l'approche d'optimisation multi-objectifs.

L'architecture globale est présentée dans la figure 5.5. Une population initiale (ensemble de compositions de services répondant au besoin) est générée. Cette population initiale a une taille fixée. Un premier front de Pareto est construit. Les individus de la population initiale qui n'appartiennent pas à ce front (composition de service non optimale) sont alors mutés à l'aide des algorithmes d'optimisation, puis ré-injectés dans la population étudiée. Le processus est ré-itéré un nombre borné de fois.

5.2.3 Services avec plusieurs entrées / sorties

Comme il est peu réaliste de ne considérer que des services avec une unique entrée et une unique sortie, nous nous sommes ensuite intéressés à des services plus généraux. La difficulté est

alors la construction du plan de réalisation. En effet, il devient alors très difficile de construire le plan de réalisation à la volée. Le choix a été fait de se baser sur plusieurs plans (composés de tâches) répondant à la problématique. Nous supposons aussi connu un ensemble de services (et leur qualité de service) capables de réaliser chaque tâche.

Le même principe que précédemment est alors appliqué : création de la population initiale (affectation aléatoire d'un service à une tâche pour chaque plan), puis itération des mutations des compositions qui ne sont pas sur le front de Pareto.

5.2.4 Algorithmes d'optimisation utilisés

Deux algorithmes d'optimisation ont été utilisés. Tous les deux sont des méthodes approchées (rapides mais renvoyant une valeur proche de l'optimum et non l'optimum lui-même). Les deux algorithmes sont des algorithmes génétiques basés sur l'intelligence en essaim. Le premier s'inspire du comportement du coucou et le second du comportement des chauves-souris.

Les deux algorithmes fonctionnent sur le même principe : ils possèdent une population initiale qu'ils font évoluer par mutation. Après une phase de mutation, les meilleurs individus sont gardés, les moins bons sont oubliés et de nouveaux individus sont ajoutés (pour explorer un espace d'états le plus grand possible).

Le problème des deux algorithmes choisis est la grande part d'aléatoire qu'ils intègrent, n'aidant pas au contrôle (et à la compréhension !) des résultats. Les résultats ont donc été difficiles à analyser et peu concluants.

5.3 Bilan

Ce chapitre présente deux travaux qui s'intéressent à la qualité de service de la composition de services. Le premier ne s'intéresse qu'à un unique critère, le temps d'exécution, et s'appuie sur des algorithmes d'apprentissage automatique pour choisir une composition de services parmi un ensemble. Le second s'intéresse à des critères multiples et propose de coupler la découverte et la sélection de la composition de services.

Les algorithmes d'apprentissage automatique se sont révélés efficaces pour réaliser le choix entre plusieurs compositions de services. Nous avons fait nos expériences sur une seule machine, ce qui explique que la régression linéaire se comporte bien. En effet, même si les modèles de performance sont difficiles à construire à partir de la complexité théorique des services, il n'est pas étonnant que le temps d'exécution soit très fortement corrélé au nombre de services appelés, à la taille des matrices avec lesquelles ils sont appelés et à leur complexité théorique, qui sont les informations présentes dans l'espace de caractéristiques.

Il faudrait étendre ce travail en regardant si les résultats sont toujours aussi bons dans le cas de compositions exécutées sur des machines plus complexes (multi-cœurs, plusieurs machines, exécutions parallèles...). Il faudrait également vérifier que ces travaux passent à l'échelle avec

un nombre plus important de services disponibles. Et enfin, il faudrait s'intéresser à d'autres domaines d'applications. Pour utiliser ces travaux dans un autre domaine, la seule modification à apporter est la redéfinition de l'espace de caractéristiques. En effet, celui présenté ici est fortement lié à l'algèbre linéaire. Le travail de définition de cet espace de caractéristiques n'est pas simple et est primordial pour le bon comportement des algorithmes d'apprentissage (nous l'avons vu avec les deux espaces choisis).

L'idée de guider l'énumération des solutions vers la meilleure solution pour éviter d'explorer tout l'espace d'état est prometteuse mais, comme il a déjà été dit, la mise en œuvre n'a pas été satisfaisante. Il aurait été intéressant d'utiliser d'autres algorithmes comme ceux utilisés dans des travaux récents de recherche de produits dans une ligne de produits [HLL+16]. Par exemple, étudier le comportement de NSGA-II [DPAM02] dans notre contexte aurait été pertinent. C'est un algorithme génétique devenu un algorithme de référence pour les approches multi-objectifs s'intéressant à générer le front de Pareto.

CHAPITRE 6

CONCLUSION ET PERSPECTIVES

6.1 Conclusion

Dans ce manuscrit la formalisation et la vérification de la composition de services, au sens large, sont étudiées sous plusieurs points de vue.

Dans un premier temps, la découverte d'une composition de services guidée par la fonctionnalité souhaitée est abordée. Deux approches sont étudiées. Dans la première, le plan de la composition n'est pas connu et doit être découvert, dans la seconde le workflow de la composition est connu, et il faut affecter les services aux tâches. Usuellement, il est plus difficile de découvrir le plan de composition, mais dans l'approche proposée, la composition est obtenue "gratuitement" : il suffit de rappeler l'unificateur sur les termes complexes. Les deux études ont la particularité de se placer dans un domaine d'application précis, où les opérateurs et leurs propriétés sont bien connus et "normalisés". Les travaux s'appliqueraient à n'importe quel autre domaine applicatif possédant les mêmes caractéristiques mais ne s'appliquent pas à des domaines d'application peu structurés.

Dans un deuxième temps, la vérification d'une composition de service paramétrée par le mode d'interaction est étudiée. Pour cela, la diversité des interactions asynchrones est étudiée, une hiérarchie de modèles de communication asynchrone est présentée, et un framework de vérification de compatibilité de services est présenté. Ce framework est paramétré par le modèle de communication et par la propriété à vérifier. Cette propriété peut être en rapport avec le comportement (plus de message en transit par exemple) ou fonctionnelle (les services terminent par exemple). Ces travaux présentent un double intérêt : présenter la multitude du monde asynchrone (aussi bien avec les diverses propriétés d'ordre de délivrance, qu'avec les propriétés applicatives) là où elle est souvent réduite au dual du monde synchrone, et proposer un outil

automatisé de vérification. Lorsqu'il est question du monde distribué et asynchrone, la notion de défaillance émerge naturellement. La limite des travaux actuels est de ne pas les prendre en considération.

Finalement, dans un troisième temps, la sélection de la meilleure composition de service est abordée, en utilisant la qualité de service comme critère de sélection. Deux approches sont proposées. La première consiste à construire l'ensemble des compositions possibles puis de sélectionner la meilleure composition à l'aide d'algorithmes d'apprentissage. La seconde consiste à guider la construction de la composition par la qualité de service, pour ne construire que les meilleures compositions. Ceci est réalisé à l'aide d'algorithmes d'optimisation. La première approche montre des résultats encourageants, mais possède des limites : il faut construire l'ensemble des compositions, ce qui peut être coûteux. Comme pour le courtier, l'approche basée apprentissage automatique donne de bons résultats car le domaine d'application des compositions est restreint et est un sous-ensemble des mathématiques. La seconde approche, qui consiste à guider la construction de la composition par le critère de choix de la composition, permet de ne pas construire toutes les compositions dans le but d'en choisir une unique et mériterait d'être étudiée de façon plus poussée.

6.2 Perspectives

L'ensemble de mes travaux portent sur l'usage de la modélisation et de la vérification formelle. La première phase est alors la formalisation du domaine d'application / des problèmes. Dans mes travaux précédents, différents formalismes ont été utilisés : types abstraits algébriques, workflows, systèmes de transitions... Cette formalisation a un impact important sur la façon dont pourront être prouvés les résultats : soit des outils existent déjà pour raisonner avec ces formalismes et ils seront utilisés pour répondre à notre problème (algorithme d'apprentissage automatique, algorithme d'optimisation, TLC / TLAPS, Rodin), soit les outils devront être développés (courtier, framework de vérification de compatibilité). Il est alors indispensable d'avoir des garanties sur ces outils, leurs sorties doivent donc être prouvées (correction et complétude du courtier et du framework de vérification de compatibilité).

Mes travaux vont continuer de s'articuler autour des problématiques de formalisation, et des besoins en outils et méthodes associés à ces formalisations.

6.2.1 Modélisation

Les modélisations formelles ont pour moi deux objectifs. Le premier est de permettre une meilleure compréhension du domaine, de ce que nous manipulons. Le second est de permettre de pouvoir raisonner formellement. La modélisation et les raisonnements formels ayant un coût non négligeable, ils se restreignent souvent aux domaines critiques. Malgré leur coût, je suis pour une généralisation de ces approches formelles, car avec l'omniprésence de l'informatique

dans notre quotidien, tout domaine est en train de devenir critique. C'est pour cette raison que je ne me restreindrai pas à des domaines dits critiques, mais que j'étudierai tout domaine dont la complexité appelle à la rigueur.

6.2.1.1 Les défaillances dans les systèmes répartis

Ces travaux ont débuté et correspondent aux travaux de thèse d'Adam Shimi (Oct 2017 -). Thèse financée par le projet ANR PARDI et co-encadrée avec Philippe Quéinnec.

Les compositions de services font interagir un grand nombre (parfois non connu) de services. Avec la multiplication des services, les défaillances ne sont plus un cas isolé et rare, mais sont inévitables et doivent être prises en compte dans le développement et la vérification formelle d'un système. Dans le cas de systèmes communiquant par messages, les défaillances peuvent être de diverses natures : arrêt définitif d'un site, arrêt temporaire d'un site, perte de message, duplication de message, altération de message... Plutôt que de formaliser chaque structure d'algorithme et chaque type de défaillance, l'idée est de proposer un cadre général de vérification de système paramétré par les défaillances, à l'image du framework de la section 4.3 qui lui était paramétré par le modèle de communication.

Modèles par tour Pour pouvoir modéliser la structure des algorithmes, nous allons nous restreindre aux algorithmes fonctionnant par tours : à chaque tour, les pairs diffusent un message, reçoivent des messages, calculent leur nouvel état et passent au tour suivant. Ce modèle a l'avantage, tout en restreignant les comportements, de garder une grande généralité et permet de décrire bon nombre d'algorithmes. L'utilisation des tours dans les algorithmes par passage de message n'est pas récente. Dans [AFL81] les tours sont utilisés comme une abstraction de la complexité temporelle. Les tours sont également utilisés pour représenter des réseaux dynamiques [KO11] : chaque tour correspond à un graphe de communication fixé, l'aspect dynamique étant induit par la possibilité de changer le graphe entre les tours. Finalement, beaucoup d'algorithmes tolérants aux fautes sont structurés en tours. Nous pouvons citer le consensus avec au plus f défaillances [FL82], ou la version asynchrone utilisant des détecteurs de défaillances pour une minorité de défaillances [CHT96].

Modélisation des défaillances avec le modèle Heard-Of Pour pouvoir modéliser les différentes défaillances, nous allons utiliser le modèle Heard-Of [CBS09] qui abstrait celles-ci en représentant par un prédicat l'ensemble des pairs pour lesquels un pair donné reçoit un message à un tour donné. Cela permet de modéliser à la fois les pertes de messages, les arrêts définitifs ou temporaires par exemple, mais également l'aspect dynamique d'un réseau de communication.

Ce modèle est à la fois endogène et exogène : soit le prédicat impose les messages reçus par chaque pair dans chaque tour, soit les collections Heard-Of sont observées *a posteriori* et il s'agit de raisonner dessus.

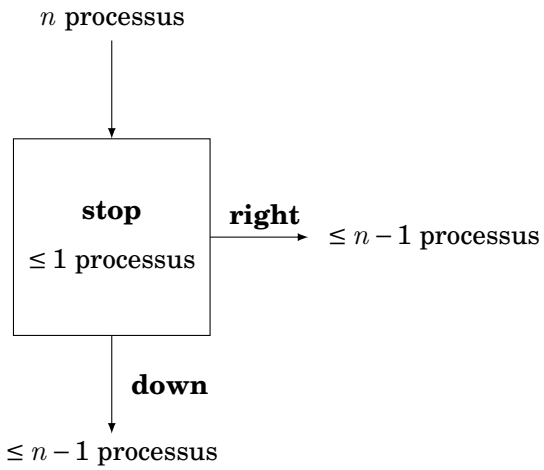
Problématique à court terme : Nous avons l'habitude de manipuler les défaillances avec des modèles de plus bas niveau que le modèle Heard-Of. L'objectif est d'étudier le modèle Heard-Of pour mieux comprendre son lien avec les modèles de défaillances traditionnels et ainsi apprendre à raisonner avec ce modèle.

6.2.1.2 Les systèmes non bloquants

Actuellement, le gain de performance des ordinateurs n'est obtenu que par l'augmentation du degré de parallélisme : la fréquence des horloges n'augmente plus, et les architectures multi-processeurs multi-cœurs sont largement répandues dans les ordinateurs grand public comme pour le calcul intensif. Pour autant, l'exploitation du parallélisme de ces architectures n'est pas évidente et son utilisation dans des systèmes critiques (embarqués notamment) reste limitée. Les systèmes concurrents à mémoire partagée sont des systèmes complexes, tant à développer qu'à appréhender, et difficiles à mettre au point. L'incertitude de leur fiabilité est un souci majeur pour les systèmes embarqués comme pour la sécurité.

Les problèmes de synchronisation et de coordination sont usuellement résolus avec des verrous logiciels, mais l'on sait que la défaillance d'un programme possédant des verrous bloque définitivement la progression des autres programmes, sauf à adjoindre de complexes mécanismes de reprise sur défaillance. Or, les algorithmes non bloquants outrepassent cette limitation en offrant des opérations de synchronisation et de partage de données dans lesquels un processus donné n'est jamais empêché de progresser, quelle que soit la progression des autres processus. La vitesse de progression d'un processus donné est alors indépendante des autres processus. Bien que connus et étudiés depuis plusieurs années, ces algorithmes, d'un intérêt pratique majeur, restent peu utilisés.

Les algorithmes non bloquants s'appuient sur certaines instructions processeur aux propriétés bien définies (les registres, cases mémoire à lecture et écriture atomiques y compris en multiprocesseurs, opérations test-and-set et compare-and-swap de lecture et modification atomiques...). L'écriture et la compréhension de tels programmes sont délicates et sujettes à erreurs. Les preuves de correction écrites manuellement en pseudo langage mathématique laissent souvent à désirer, ou tout au moins, ne permettent pas de certifier les algorithmes corrects. En fait, ces algorithmes exhibent un nombre considérable d'états accessibles, et pour chaque état, de multiples transitions possibles, alors que les algorithmes auxquels l'informaticien est habituellement confronté sont largement séquentiels avec quelques rares branchements. Les algorithmes non bloquants sont également difficiles à tester du fait de l'explosion combinatoire des entrelacements possibles dus au parallélisme. Ainsi, des algorithmes non bloquants élémen-




```
direction(id)
  X := id
  if Y
  then dir := right
  else Y := true
    if (X=id)
    then dir := stop
    else dir := down
    endif
  endif
return dir
```

FIGURE 6.1 – Spécification et implantation du splitter

taires ont été introduits dans la bibliothèque Java (classe ConcurrentLinkedQueue par exemple) mais les premières implantations se sont révélées erronées, exposant l'utilisateur à de possibles pertes de données mettant en péril la sécurité de l'application.

À mon sens, ces algorithmes nécessitent impérativement une preuve formelle mécanisée. D'importants progrès sont actuellement en cours dans le domaine de la vérification assistée ou automatisée de systèmes paramétrés, où le nombre d'entités concurrentes est un paramètre non explicité. Ces avancées permettent de vérifier un algorithme pour une classe d'environnements et pour un degré arbitrairement grand de parallélisme. La mise en œuvre de ces méthodes de vérification paramétrée dans le contexte des algorithmes non bloquants est un problème ouvert. Le préliminaire à cette vérification est de disposer d'une modélisation formelle des algorithmes, et par conséquent une description rigoureuse et mécanisable des mécanismes processeur impliqués.

Étude préliminaire : le splitter Le splitter [MA95] est une brique de base d'algorithmes non bloquants. Il permet de séparer des processus. Sa spécification est décrite dans la figure 6.1. x processus appellent concurremment (ou pas) le splitter. Les contraintes sont :

- au plus un processus termine avec $stop$;
- s'il y a uniquement un processus, il termine avec $stop$;
- au plus $(x-1)$ processus terminent avec $right$;
- au plus $(x-1)$ processus terminent avec $down$.

La figure 6.1 présente également un algorithme simple qui implante le splitter à l'aide de deux registres partagés X et Y.

Même sur un exemple aussi simple que le splitter, la modélisation du problème n'est pas triviale. L'algorithme seul ne fournit pas toutes les indications. En effet, cet algorithme décrit

le comportement séquentiel d'un processus en isolation. Hors, caché derrière le fait que X et Y sont deux registres partagés se cachent des informations importantes sur la granularité de l'entrelacement des actions des différents processus.

Problématique à court terme : Modéliser les systèmes non bloquants dans toute leur subtilité.

6.2.1.3 Les systèmes interactifs

Un autre domaine d'étude concerne les systèmes interactifs et sera abordé avec l'équipe Informatique Interactive de l'ENAC. Ce ne sera pas le cœur de mes travaux futurs, mais elle se place dans la continuité des travaux sur la qualité de service. Dans les travaux précédents, les informations de qualité de service sont données *a priori*, il s'agit ici de modéliser les systèmes interactifs pour pouvoir calculer une propriété de qualité de service : la latence.

Un logiciel interactif est une application informatique qui réagit, tout au long de son exécution, à différentes sources d'événements, en produisant notamment une représentation perceptible de son état interne [BL04, MR92]. Or, l'utilisabilité d'une application interactive peut être sensiblement impactée par les délais de propagation de l'information et par ses variations, c'est-à-dire la latence et la gigue. Le problème est frappant pour des applications exploitant des interfaces tactiles ou de réalité augmentée, où des décalages entre interaction et représentation peuvent rendre le système inutilisable [PS11, MW93, WB94]. Pour autant, la latence est bien souvent une caractéristique qui n'est prise en compte que très tardivement dans les processus de développement, généralement par des mesures expérimentales a posteriori, lorsque le système est déjà achevé. C'est le cas par exemple dans les systèmes du contrôle aérien (contrôle radar, remote tower) où la latence pour la mise à jour de l'affichage des vols est évaluée expérimentalement afin de dimensionner les limites d'espacement entre les aéronefs. La latence globale du système a donc des conséquences directes sur les capacités de gestion du trafic aérien [CDH16]. Les sources de latence dans un système interactif sont multi-factorielles. Dans une vision en couche d'un système, on peut l'attribuer au matériel (propagation des signaux physiques), au système d'exploitation (interruption, priorités, noyau), puis à l'empilement plus ou moins opportun de couches logicielles jusqu'à l'application elle-même.

L'objectif est de permettre au concepteur et au développeur de logiciel interactif la prise en compte de la latence et de ses effets dans l'ensemble des phases du cycle de vie du logiciel. Pour cela la première étape est de modéliser le phénomène de latence afin d'identifier les parties critiques des systèmes interactifs.

Problématique à court terme : Modéliser les système interactifs et leur différentes couches pour évaluer leur qualité de service.

6.2.1.4 Bilan

La chance de l'informatique est d'être une discipline scientifique avec des fondements théoriques qui devraient permettre de donner des garanties sur les programmes informatiques. Bien sur ces garanties ne sont pas gratuites, mais cela ne pardonne pas la situation actuelle. Personne ne s'émeut d'une application, d'un ordinateur ou d'un téléphone "qui buggue". L'utilisateur relance son application / téléphone / ordinateur et recommence. Tant que l'utilisateur s'accommodera d'une telle situation, les mœurs n'évolueront pas. Avec la multiplication des calculateurs dans nos vies et la montée en puissance de l'Internet des objets, il est fort à parier que la situation ne devienne plus acceptable d'ici quelques années.

Pour moi, une des clés pour réduire les erreurs de spécification / conception / programmation (car un bug est par définition une erreur humaine) est une meilleure compréhension des objets informatiques que nous manipulons, et cela passe donc par une modélisation fine des domaines et applications (la preuve venant en bonus et sera discutée dans la section suivante). Les trois domaines d'applications précédents ont permis d'illustrer la grande diversité des problèmes de modélisation. Si, pour certains domaines, des idées de formalisation sont connues, pour d'autres les possibilités restent diverses. Bon nombre de descriptions formelles existent et peuvent être réutilisées (notamment pour utiliser les outils associés), mais il peut également être nécessaire d'introduire notre propre formalisme pour répondre à un problème non abordé formellement jusque-là.

Plusieurs pistes sont envisagées pour aider à développer la propagation des modélisations formelles dans le développement informatique. D'abord, il faut que les formalisations ne se restreignent pas uniquement aux systèmes critiques. Leur démocratisation passera par le fait qu'on remette en cause leur nécessité uniquement dans ces domaines. Comme je l'ai déjà dit l'omniprésence de l'informatique rend presque tous les domaines critiques. Plus les domaines d'applications seront variés, plus la banque de données des modélisations formelles sera fournie et meilleures seront les chances de toucher d'autres domaines. Cette diversité des modélisations formelles est aussi un point clé. Pour amorcer l'"effet boule de neige", il faut commencer par donner l'exemple. Bien évidemment, quels que soient les domaines d'application de mes travaux, je m'efforcerai de modéliser formellement mon problème, soit en réutilisant des formalismes existants, soit en en proposant de nouveaux. Finalement, il faut sensibiliser les collègues mais aussi, et surtout, les étudiants à ces problèmes.

Problématique à long terme : Aider à la généralisation des modélisations formelles.

6.2.2 Outils et méthodes

L'objectif d'une bonne formalisation, en plus d'apporter une meilleure compréhension du domaine, est de permettre de raisonner formellement et / ou proposer des outils manipulant cette formalisation. Par exemple la formalisation de l'algèbre linéaire à l'aide de types abstraits

algébriques a permis de proposer un courtier, permettant de trouver la composition de services répondant à un besoin.

6.2.2.1 Les défaillances dans les systèmes répartis

Il est intéressant de paramétrer les algorithmes répartis tolérants aux fautes par, au moins, le nombre de pairs et le nombre de défaillances qu'ils peuvent supporter. Le but est alors de prouver que le système est correct quel que soit le nombre de processus et de défaillances ou pour une relation donnée entre le nombre de processus et de défaillances. Concernant les défaillances, plusieurs résultats d'impossibilité sont connus aussi bien pour les systèmes communiquant par message que pour les systèmes à mémoire partagée [FLP85, Her88, CHTCB96, TM96]. Les détecteurs de défaillances [CT96, DFKM97, GHK+07] ont été introduits pour contourner ces résultats et il a été montré que des implantations réalistes sont possibles (par exemple [ACT99]). Des blocs génériques (registres atomiques, consensus...) peuvent être construits à partir des détecteurs de défaillances et utilisés pour développer des systèmes plus complexes. Malgré cette formalisation des détecteurs de défaillances, les preuves des algorithmes distribués sont généralement réalisées pour un modèle spécifique, et souvent ni les défaillances possibles ni le détecteur de défaillances utilisé n'est vu comme un paramètre du système.

Une autre approche possible pour traiter les défaillances est le modèle Heard-Of présenté précédemment. Son lien avec les détecteurs de défaillances sera étudié. Cela permettra de mieux saisir les subtilités du modèle Heard-Of mais également ses limites. S'il n'a pas la même expressivité que les détecteurs de défaillances, quels sont les systèmes que nous pouvons vérifier avec l'une des approches mais pas avec l'autre ?

Le modèle Heard-Of permet d'évacuer le problème de la multiplicité des causes de défaillances en les remplaçant par la manipulation d'un prédicat. Cela conduit à une uniformisation des représentations des défaillances qui permet de réaliser des preuves en évacuant la raison du mauvais comportement et en ce concentrant uniquement sur la conséquence de celui-ci. En effet peu importe qu'un message ne soit pas arrivé car un site ne l'a pas envoyé (arrêt du site) ou car le réseau l'a perdu (réseau non fiable), ce qui compte c'est que le système fonctionne correctement, même si le message n'est pas reçu.

La modélisation des défaillances sous forme de prédicat logique est également un atout pour la réalisation des preuves. En effet, cette expression sous forme logique est manipulable naturellement dans les preuves et devrait faciliter leur écriture. De plus, lors de l'écriture de ces preuves, il devrait être possible d'identifier les propriétés nécessaires, sur le prédicat Heard-Of, pour la réalisation de la preuve. Cela permettrait de générer un prédicat "minimal" (pour l'implication) qui garantit la correction d'un système.

Problématique à court terme : Étudier comment le modèle Heard-Of nous permet de faciliter la conception et la vérification d'un système réparti.

6.2.2.2 Les systèmes non bloquants

La preuve des algorithmes non bloquants n'est pas triviale, mais pourtant nécessaire car la complexité des entrelacements possibles les rendent difficiles à comprendre.

Étude préliminaire : le splitter Reprenons le splitter décrit précédemment. L'intuition des preuves de correction est assez simple :

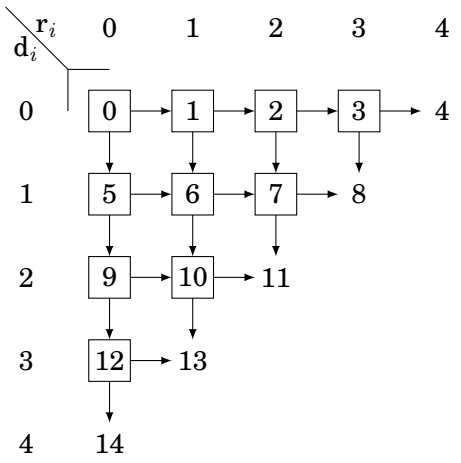
- Au plus $(x-1)$ processus terminent avec $right$: les processus obtenant $right$ trouvent Y, qui a nécessairement été positionné par un processus obtenant $down$ ou $stop$.
- Au plus $(x-1)$ processus terminent avec $down$: soit p_i le dernier processus ayant écrit X. Si p_i trouve Y, il obtiendra $right$. Sinon son test $X = id_i$ lui fera obtenir $stop$.
- Au plus un processus termine avec $stop$: soit p_i le *premier* processus trouvant $X = id_i$. Alors aucun processus n'a modifié X depuis que p_i l'a fait. Donc tous les processus suivants trouveront Y et obtiendront $right$ (car p_i a positionné Y), et les processus en cours qui n'ont pas trouvé Y ont vu leur écriture de X écrasée par p_i (puisqu'elle n'a pas changé jusqu'au test par p_i). Ils ne pourront donc trouver X égal à leur identifiant et obtiendront donc $down$.

J'ai réalisé deux preuves mécanisées du splitter, l'une avec Why3 et l'autre avec TLAPS. Elles sont disponibles à l'adresse suivante : http://hurault.perso.enseeiht.fr/splitter/index.html. Notons que dans ces deux formalisations, la modélisation du système complet n'est pas totalement identique. Dans la description Why3, une boucle principale choisit le processus qui fait un pas, et la terminaison de la boucle est garantie par un nombre de pas fixé. Dans la modélisation avec TLA$^+$, je me suis appuyée sur les mécanismes de TLA$^+$ pour modéliser la concurrence et l'entrelacement. Cela fait écho à la problématique de modélisation des systèmes non bloquants.

Il faut noter que, malgré la simplicité du splitter, les invariants inductifs nécessaires aux preuves ne sont pas triviaux. Ils nécessitent de "re-raconter" l'algorithme. Le mécanisme de décomposition des obligations de preuve de Why3 a fortement guidé l'écriture de ceux-ci.

Dans le même temps des versions du splitter ont été vérifiées, par un collègue, avec TLC (le vérificateur de modèle de TLA$^+$) et SPIN. Avec Promela et SPIN, il a pu être vérifié jusqu'à 7 processus. Avec TLC, il a été vérifié jusqu'à 9 processus. Le point notable est que le nombre d'états accessibles représente environ 60% des états possibles. Ce chiffre est considérable. En effet, c'est une des particularités des algorithmes non bloquants que de pouvoir atteindre une majorité des états possibles. Ceci rend leur test plus compliqué car il est difficile de garantir une couverture de test conséquente, et la vérification de modèle explose en mémoire très rapidement.

Pour aller plus loin : le renommage Le problème du renommage consiste à vouloir renommer les processus, en garantissant un nom unique. Il est par exemple nécessaire pour réduire l'espace de nommage. La solution proposée par Moir et Anderson [MA95] est présentée dans




```
renamme(id)
  while (d+r < NP-1  /\ !stop ) do
     X[d][r] := id;
     if Y[d][r]
     then r := r + 1  \* right
     else
        Y[d][r] := TRUE
        if (X[d][r] = id)
        then stop := TRUE     \* stop
        else d := d + 1  \* down
        endif
     endif
  end while
return (NP*d) + r - (d*(d-1)) \div 2
```

FIGURE 6.2 – Implantation du renommage

la figure 6.2, où NP représente le nombre de processus et X et Y sont des variables partagées. A noter que la condition $d + r < NP - 1$ n'est pas indispensable (mais est présente dans la version originale) mais facilite les preuves, notamment sur le non-débordement des tableaux. Elle consiste à enchaîner les splitters, garantissant ainsi le fait qu'un processus ne peut pas bloquer la progression d'une autre.

Le renommage a été modélisé en TLA$^+$. TLC arrive à le prouver pour 4 processus mais pas plus (en temps raisonnable). Nous voyons ici les limites des vérificateurs de modèles sur ce genre d'algorithme.

La preuve proposée par Moir et Anderson reprend le même principe que celle que j'ai fait du splitter : plusieurs invariants inductifs, explicitant le mécanisme de l'algorithme : "si un processus est à telle étape du programme, alors un autre est à telle étape / telle variable a telle valeur / . . . ". Elle n'utilise pas les résultats sur les splitters, car ceux-ci étant appelés de façon concurrente, ils ne peuvent pas être vus comme des boîtes noires, et ils n'apparaissent pas explicitement dans le code.

Une preuve en TLAPS du renommage, en se basant sur la preuve de Moir et Anderson, est en cours de réalisation. Néanmoins, nous voyons vite les limites de ce genre de preuves mécanisées qui, même sur des exemples simples, ne sont pas évidentes. De plus, il est quand même dommage de ne pas pouvoir réutiliser les propriétés du splitter pour prouver l'algorithme de renommage. En effet, l'intuition de la preuve est triviale lorsque l'on voit l'algorithme de renommage comme une grille de splitter, comme dans la figure 6.2.

Piste d'études Comme il a été acté dans la présentation du contexte, les algorithmes non bloquants sont difficiles à tester à cause du nombre impressionnant d'entrelacement possibles

des processus. Les vérificateurs de modèle arrivent vite à leur limite à cause de l'explosion du nombre d'états. Pour garantir leur correction, la preuve est donc nécessaire.

Le souci des algorithmes non bloquants est qu'ils ne peuvent pas être définis avec des appels de hauts niveaux à des blocs plus petits et plus simples. Ceci est lié au fait qu'il faut prendre en compte tous les entrelacements possibles, et que l'atomicité des actions est réduite à la lecture / écriture d'un registre puisque l'inexistence de verrou interdit une autre atomicité. Cette écriture non modulaire des algorithmes interdit donc une réalisation modulaire de la preuve (comme dans le cas du renommage où la preuve originale n'utilise pas les propriétés du splitter). Cela rend ces algorithmes excessivement difficiles à prouver.

Problématique à court terme : Chercher les invariants inductifs des algorithmes non bloquants ne semblant pas raisonnable, l'idée est d'étudier la possibilité de rendre les preuves modulaires en montrant que les algorithmes se réduisent à des algorithmes modulaires. Nous sommes alors confrontés à deux problèmes ouverts : cette réduction est elle possible ? Comment cette réduction peut elle-être intégrée aux outils de preuve ?

6.2.2.3 Les systèmes interactifs

Pour permettre au concepteur et au développeur de logiciels interactifs la prise en compte de la latence et de ses effets dans l'ensemble des phases du cycle de vie du logiciel, il est nécessaire de mettre au point des outils logiciels permettant la mesure, la visualisation, la spécification et la vérification de ces propriétés. Ces outils rendront possible, lors de la conception, l'évaluation objective de différentes solutions d'architecture logicielle. Dans une optique d'utilisation dans des domaines critiques, ces outils devront s'appuyer sur des méthodes formelles afin de pouvoir garantir leurs corrections.

Problématique à court terme : Proposer des outils de vérification de systèmes interactifs ainsi qu'une méthodologie de conception centrée sur la prise en compte de la latence et utilisant ces outils logiciels.

6.2.2.4 Bilan

Toujours dans l'objectif de rendre les applications informatiques plus sûres, la seconde phase naturelle après la modélisation est la preuve / la recherche de garanties sur les sorties et comportements d'une application. Bien sûr cette partie ne peut pas être indépendante de la réflexion sur la modélisation du problème et surtout des choix de modélisations réalisés. Une fois la modélisation faite, un grand pas est réalisé pour faciliter la vérification. Ceci est notamment vrai avec le développement et la plus grande efficacité des vérificateurs de modèles. Une fois que

la spécification TLA$^+$ est réalisée, il est moins coûteux de demander à TLC de vérifier le modèle que de réaliser son propre jeu de tests.

Néanmoins, nous ne sommes pas toujours dans un cas de figure où les outils sont disponibles. C'est pourquoi je continuerai à m'efforcer de proposer des outils (prouvés corrects) faciles d'utilisation pour des non-spécialistes des méthodes formelles (comme ça sera, je l'espère, le cas pour la vérification de systèmes avec fautes, et pour les systèmes interactifs).

Je parle beaucoup de preuves mécanisées ou d'outils (prouvés corrects, si possible mécaniquement) d'aide au développement. Pour moi cette mécanisation des preuves est la seule garantie de la correction de celles-ci, car elle écarte l'erreur humaine dans la réalisation de celles-ci. Qui n'a jamais publié une preuve erronée ou incomplète ? Bien sûr cette mécanisation totale n'est pas possible dans l'immédiat, et on peut discuter de sa nécessité. Mais les progrès impressionnants de ces dernières années, par rapport aux solveurs SMT par exemple, donnent bon espoir. Malheureusement le revers de la médaille de cette mécanisation est, parfois, la perte de la structure de la preuve (le SMT solveur dit oui ou non mais on ne sait pas pourquoi). Pour aider à la généralisation des preuves mécanisées, et à la facilité d'utilisation des assistants de preuve, je vais continuer à servir de cobaye pour faire remonter les besoins aux concepteurs d'outils de preuves (comme c'est le cas dans le projet ANR PARDI). Je ne développe pas des assistants de preuve ou des vérificateurs de modèles, mais en tant qu'utilisatrice, je peux aider à améliorer leur facilité d'utilisation pour espérer une démocratisation. L'étude poussée des systèmes non bloquants amènera probablement une réflexion de plus haut niveau sur l'intégration des techniques de preuves nécessaires dans les outils de preuve.

Problématique à long terme : Aider à la généralisation des preuves (mécanisées) formelles.

BIBLIOGRAPHIE

[AAB+01] Dorian Arnold, Sudesh Agrawal, Susan Blackford, Jack Dongarra, Michelle Miller, Kiran Sagi, Zhiao Shi, and Sathish Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001.

[AAM17] Pascal André, Christian Attiogbé, and Jean-Marie Mottu. Combining techniques to verify service-based components. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2017, Porto, Portugal, February 19-21, 2017.*, pages 645–656. SciTePress, 2017.

[ABB+99] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, Jim Demmel, Jack Dongarra, Jeremey Du Croz, Sven Hammarling, Anne Greenbaum, Alan McKenney, and Danny Sorensen. *LAPACK Users' guide (third ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[ABD+05] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job submission description language (JSDL) specification, version 1.0. Technical report, Global Grid Forum, 2005.

[Abr96] Jean-Raymond Abrial. *The B-book : assigning programs to meanings*. Cambridge University Press, New York, NY, USA, 1996.

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[ACKM10] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services : Concepts, Architectures and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[ACT99] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theor. Comput. Sci.*, 220(1) :3–30, June 1999.

[ADC96] María Virginia Aponte and Roberto Di Cosmo. Type isomorphisms for module signatures. In Herbert Kuchen and S. Doaitse Swierstra, editors, *Programming*

Languages : Implementations, Logics, and Programs, pages 334–346, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[AEY01] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of msc graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, ICALP '01, pages 797–808, London, UK, UK, 2001. Springer-Verlag.

[AFL81] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. A difference in efficiency between synchronous and asynchronous systems. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 128–132, New York, NY, USA, 1981. ACM.

[AG10] Nick Antonopoulos and Lee Gillam. *Cloud Computing : Principles, Systems and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5) :672–713, September 2002.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, ICALP '89, pages 1–17, London, UK, UK, 1989. Springer-Verlag.

[ASS+08] Hrachya V. Astsatryan, Vladimir Sahakyan, Yuri Shoukouryan, Michel J. Daydé, Aurélie Hurault, Marc Pantel, and Eddy Caron. A grid-aware web portal with advanced service trading for linear algebra calculations. In *High Performance Computing for Computational Science - VECPAR 2008, 8th International Conference, Toulouse, France, June 24-27, 2008. Revised Selected Papers*, volume 5336, pages 150–159. Springer, 2008.

[ASS+11] Hrachya Astsatryan, Vladimir Sahakyan, Youri Shoukouryan, Myasnik Srapyan, Michel Daydé, Aurélie Hurault, and Romulus Grigoras. Introduction of a Grid-Aware Portlet for Numerical Calculations (regular paper). In *International Conference On Parallel, Distributed and Grid Computing (PDGC), Jaypee University of Information Technology Waknaghat, Solan, H.P., India, 28/10/2010-30/10/2010*, pages 67–70, http ://ieeexplore.ieee.org/, janvier 2011. IEEExplore digital library.

[ASS+12] Hrachya Astsatryan, Vladimir Sahakyan, Youri Shoukouryan, Michel Daydé, and Aurélie Hurault. Enabling Large-Scale Linear Systems of Equations on Hybrid HPC Infrastructures (regular paper). In *ICT Innovations 2011, Skopje, 14/09/2011-16/09/2011*, volume 150 of *Advances in Intelligent and Soft Computing*, pages 239–245. Springer, 2012.

[ASS+13] Hrachya Astsatryan, Vladimir Sahakyan, Youri Shoukouryan, Michel Daydé, Aurélie Hurault, Ronan Guivarch, Harutyun Terzyan, and Levon Hovhannisyan. On the Easy Use of Scientific Computing Services for Large Scale Linear Algebra and Parallel Decision Making with the P-Grade Portal. *Journal of Grid Computing*, 11(2) :239–248, juin 2013.

[AT85] Stefan Arnborg and Erik Tidén. Unification problems with one-sided distributivity. In *Proc. of the first international conference on Rewriting techniques and applications*, pages 398–406, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[AvH03] Grigoris Antoniou and Frank van Harmelen. Web Ontology Language : OWL. In *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2003.

[BBG11] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[BBGB16] Srividya Bansal, Ajay Bansal, Gopal Gupta, and M. Brian Blake. Generalized semantic web service composition. *Serv. Oriented Comput. Appl.*, 10(2) :111–133, June 2016.

[BBO12] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *39th Symposium on Principles of Programming Languages*, POPL '12, pages 191–202. ACM, 2012.

[BCH15] Serial Rayene Boussalia, Allaoua Chaoui, and Aurélie Hurault. Qos-based web services composition optimization with an extended bat inspired algorithm. In *Information and Software Technologies - 21st International Conference, ICIST 2015, Druskininkai, Lithuania, October 15-16, 2015, Proceedings*, volume 538 of *Communications in Computer and Information Science*, pages 306–319. Springer, 2015.

[BCH+16] Serial Rayene Boussalia, Allaoua Chaoui, Aurélie Hurault, Meriem Ouederni, and Philippe Quéinnec. Multi-objective quantum inspired cuckoo search algorithm and multi-objective bat inspired algorithm for the web service composition problem. *IJISTA*, 15(2) :95–126, 2016.

[BDD+02] Susan Blackford, Jim Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, Mike Heroux, Linda Kaufman, A. Lumsdaine, Andrew Petitet, Roldan Pozo, Karin Remington, and Clint Whaley. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Softw.*, 28(2) :135–151, 2002.

[Bel08] Michael Bell. *Service-Oriented Modeling : Service Analysis, Design, and Architecture*. Wiley Publishing, 2008.

[BK88] Ralph-Johan Back and Reino Kurki-Suonio. Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4) :513–554, 1988.

[BKK+98] Peter Borovansky, Claude Kirchner, Helène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An Overview of ELAN, 1998.

[BL04] Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 15–22, New York, NY, USA, 2004. ACM.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, United Kingdom, 1998.

[BS96] Franz Baader and Klaus Schulz. Unification in the union of disjoint equational theories : combining decision procedures. *Journal of Symbolic Computation*, 21(2) :211–243, 1996.

[BS01] Franz Baader and Wayne Snyder. Unification theory. In *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

[BSBM04] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are two web services compatible ? In *Technologies for E-Services, 5th International Workshop, TES 2004, Toronto, Canada, August 29-30, 2004, Revised Selected Papers*, volume 3324 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2004.

[BZ09] Mario Bravetti and Gianluigi Zavattaro. *Contract Compliance and Choreography Conformance in the Presence of Message Queues*, pages 37–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[CBMT96] Bernadette Charron-Bost, Friedemann Mattern, and Gerard Tel. Synchronous, asynchronous, and causally ordered communication. *Distributed Computing*, 9(4) :173–191, February 1996.

[CBS09] Bernadette Charron-Bost and André Schiper. The Heard-Of model : computing in distributed systems with benign faults. *Distributed Computing*, 22(1) :49–71, Apr 2009.

[CCG+13] Frédéric Camillo, Eddy Caron, Ronan Guivarch, Aurélie Hurault, Christian Klein, and Christian Pérez. Resource Management Architecture for Fair Scheduling of Optional Computations (regular paper). In *3PGCIC - Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Compiegne, 28/10/2014-30/10/2014*, pages 113–120. IEEE, 2013.

[CDD+05] Eddy Caron, Frédéric Desprez, Michel Daydé, Aurélie Hurault, and Marc Pantel. On deploying scientific software within the grid-tlse project. *Computing Letters (CoLe)*, 1(3) :1–5, 2005.

[CDE+00] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. *A Maude Tutorial*. SRI International, 2000.

[CDE+03] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The maude 2.0 system. In *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag, June 2003.

[CDH16] Maxime Cordeil, Tim Dwyer, and Christophe Hurter. *Immersive solutions for future air traffic control and management*, pages 25–31. Association for Computing Machinery (ACM), 11 2016.

[CDL+02] Eddy Caron, Frédéric Desprez, Frédéric Lombard, Jean-Marc Nicod, Martin Quinson, and Frédéric Suter. A scalable approach to network enabled servers. In *Proceedings of the 8th International EuroPar Conference*, volume 2400 of *Lecture Notes in Computer Science*, pages 907–910, Paderborn, Germany, August 2002. Springer-Verlag.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1981. Springer-Verlag.

[Che17] Florent Chevrou. *Formalisation of Asynchronous Interaction*. PhD thesis, National Polytechnic Institute of Toulouse, France, 2017.

[CHM+15] Florent Chevrou, Aurélie Hurault, Philippe Mauran, Meriem Ouederni, Philippe Quéinnec, and Xavier Thirioux. La composition de services dans le monde asynchrone Formalisation et vérification en TLA+ (short paper). In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL), Bordeaux, 09/06/15-10/06/15*, pages 34–39, http ://gdr-gpl.cnrs.fr, juin 2015. CNRS - GDR GPL.

[CHMQ16] Florent Chevrou, Aurélie Hurault, Philippe Mauran, and Philippe Quéinnec. Mechanized Refinement of Communication Models with TLA+. In *5th Intl. Conf. Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2016)*, pages 312–318. Springer-Verlag, May 2016.

[CHQ15] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. Automated verification of asynchronous communicating systems with TLA+. *ECEASST*, 72 : Proceedings of the 15th International Workshop on Automated Verification of Critical Systems (AVoCS 2015) :1–15, 2015.

[CHQ16] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. *Formal Asp. Comput.*, 28(5) :847–879, 2016.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4) :685–722, July 1996.

[CHTCB96] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 322–330, New York, NY, USA, 1996. ACM.

[CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design : A Foundation*. Addison-Wesley, 1988.

[CMU04] Evelyne Contejean, Claude Marché, and Xavier Urbain. CiME3, 2004. Available at http://cime.lri.fr/.

[Con04] Evelyne Contejean. A certified AC matching algorithm. In *15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 70–84, Aachen, Germany, June 2004. Springer-Verlag.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2) :225–267, March 1996.

[DDDH90] Jack Dongarra, Jeremy Du Croz, Iain Duff, and Sven Hammarling. Algorithm 679. a Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16 :1–17, 1990.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17 :279–301, 1982.

[DFKM97] Danny Dolev, Roy Friedman, Idit Keidar, and Dahlia Malkhi. Failure detectors in omission failure environments. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '97, pages 286–, New York, NY, USA, 1997. ACM.

[DIE] DIET. http://graal.ens-lyon.fr/DIET.

[Dij83] Edsger W. Dijkstra. EWD851b – reducing control traffic in a distributed implementation of mutual exclusion, 1983.

[DMS92] Nachum Dershowitz, Subrata Mitra, and G. Sivakumar. Decidable Matching for Convergent Systems. In *Proceedings of the Eleventh Conference on Automated Deduction (Saratoga Springs, NY)*, volume 607, pages 589–602, Berlin, 1992. Springer-Verlag.

[DMT99] Distributed Management Task Force. *Common Information Model (CIM) Specification*, Jun 1999.

[Dom91] Eric Domenjoud. *Outils pour la déduction automatique dans les théories associatives-commutatives*. PhD thesis, Université Nancy I, 1991.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *Trans. Evol. Comp*, 6(2) :182–197, April 2002.

[EBMN12] M. Eslamichalandar, K. Barkaoui, and H. R. Motahari-Nezhad. Service composition adaptation : An overview. In *2012 Second International Workshop on Advanced Information Systems for Enterprises*, pages 20–27, Nov 2012.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" Revisited : On Branching Versus Linear Time Temporal Logic. *J. ACM*, 33(1) :151–178, January 1986.

[FGNC01] Gilles Fedak, Cécile Germain, Vincent Neri, and Franl Cappello. XtremWeb : A Generic Global Computing System. In *CCGRID '01 : Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2001.

[FH86] François Fages and Gérard Huet. Complete Sets of Unifiers and Matchers in Equational Theories. *Theoretical Computer Science*, 43(2-3) :189–200, 1986.

[Fid89] Colin J Fidge. *Dynamic analysis of event orderings in message-passing systems*. PhD thesis, The Australian National University, 1989.

[FK97] Ian Foster and Carl Kesselman. Globus : A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2) :115–128, Summer 1997.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4) :183–186, 1982.

[FLM+05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object-oriented Systems*. Springer, New York, 2005.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2) :374–382, April 1985.

[FSYH03] Yoshinari Fukui, Andrew Stubbings, Takashi Yamazaki, and Ryutaro Himeno. Constructing a virtual laboratory on the internet : The itbl portal. In *ISHPC*, volume 2858 of *Lecture Notes in Computer Science*, pages 288–297. Springer, 2003.

[GAJG16] Ikbel Guidara, Imane Al Jaouhari, and Nawal Guermouche. Dynamic Selection for Service Composition Based on Temporal and QoS Constraints. In *International Conference on Services Computing*, Services Computing (SCC), 2016 IEEE International Conference on, San Francisco, United States, June 2016.

[GGM99] Jean-Marie Geib, Christophe Gransart, and Philippe Merle. *CORBA : des concepts à la pratique*. DUNOD, 1999.

[GHK+07] Rachid Guerraoui, Maurice Herlihy, Petr Kouznetsov, Nancy Lynch, and Calvin Newport. On the weakest failure detector ever. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 235–243, New York, NY, USA, 2007. ACM.

[GM92] Joseph Goguen and José Meseguer. Order-sorted algebra I : equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theor. Comput. Sci.*, 105(2) :217–273, 1992.

[GPSY16] Matthias Güdemann, Pascal Poizat, Gwen Salaün, and Lina Ye. Verchor : A framework for the design and verification of choreographies. *IEEE Trans. Services Computing*, 9(4) :647–660, 2016.

[Gre15] Samuel Greengard. *The Internet of Things*. The MIT Press, 2015.

[Gro94] Object Management Group. *Common Object Services Specification : Atandt/NCR, Bnr Europe Limited, Digital Equipment Corporation*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[GS89] Jean Gallier and Wayne Snyder. Complete Sets of Transformations for General E-Unification. *Theor. Comput. Sci.*, 67(2-3) :203–260, 1989.

[Han11] J.S. Hansen. *Gnu Octave : Beginner's Guide. Jesper Schmidt Hansen*. Learn by doing : less theory, more results. Packt, 2011.

[HBC15] Aurélie Hurault, Kyungim Baek, and Henri Casanova. Selecting linear algebra kernel composition using response time prediction. *Software : Practice and Experience*, 45(12) :1659–1676, 2015.

[Hei01] *Component-based Software Engineering : Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Her88] Maurice P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 276–290, New York, NY, USA, 1988. ACM.

[HLL+16] Robert M. Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip : Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Trans. Softw. Eng. Methodol.*, 25(2) :17 :1–17 :39, April 2016.

[Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10) :576–580, 1969.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8) :666–677, August 1978.

[Hoa83] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 26(1) :53–56, 1983.

[Hol97] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5) :279–295, May 1997.

[Hur06] Aurélie Hurault. *Courtage sémantique de services de calcul*. Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, décembre 2006. (Soutenance le 04/12/2006).

[HY10] Aurélie Hurault and Asim YarKhan. Intelligent service trading and brokering for distributed network services in gridsolve. In *High Performance Computing for Computational Science - VECPAR 2010 - 9th International conference, Berkeley, CA, USA, June 22-25, 2010, Revised Selected Papers*, volume 6449, pages 340–351. Springer, 2010.

[Jaf90] Joxan Jaffar. Minimal and complete word unification. *J. ACM*, 37(1) :47–85, 1990.

[JRGM05] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation in web service compositions. In *2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 181–185, March 2005.

[KB70] Donald Knuth and Peter Bendix. Simple Word Problems in Universal Algebra. In *Computational Word Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KFNM04] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The protégé owl plugin : An open development environment for semantic web applications. In *Third International Semantic Web Conference - ISWC 2004*, Hiroshima, Japan, 2004.

[KL89] Michael Kifer and Georg Lausen. F-logic : A higher-order language for reasoning about objects, inheritance, and scheme. *SIGMOD Rec.*, 18(2) :134–146, June 1989.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4) :741–843, July 1995.

[KO11] Fabian Kuhn and Rotem Oshman. Dynamic networks : Models and algorithms. *SIGACT News*, 42(1) :82–96, March 2011.

[KP06] Raman Kazhamiakin and Marco Pistore. Analysis of realizability conditions for web service choreographies. In *Proceedings of the 26th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, FORTE'06, pages 61–76, Berlin, Heidelberg, 2006. Springer-Verlag.

[KS11] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing : Principles, Algorithms, and Systems*. Cambridge University Press, March 2011.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565, 1978.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3) :872–923, 1994.

[Lam02] Leslie Lamport. *Specifying Systems*. Addison Wesley, 2002.

[LD90] R. Lalement and M. Demazure. *Logique, réduction, résolution*. Études et recherches en informatique. Masson, 1990.

[LDSY08] Yinan Li, Jack Dongarra, Keith Seymour, and Asim YarKhan. Request Sequencing : Enabling Workflow for Efficient Problem Solving in GridSolve. *International Conference on Grid and Cooperative Computing (GCC 2008)*, pages 449–458, Oct 2008.

[LYD+13] Yinan Li, Asim Yarkhan, Jack Dongarra, Keith Seymour, and Aurélie Hurault. Enabling Workflows in GridSolve : Request Sequencing and Service Trading. *Journal of Supercomputing*, 64(3) :1133–1152, juin 2013.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[MA95] Mark Moir and James H. Anderson. Wait-free algorithms for fast, long-lived renaming. *Sci. Comput. Program.*, 25(1) :1–39, 1995.

[MA04] Brandon Morel and Perry Alexander. Spartacas automating component reuse and adaptation. *IEEE Trans. Softw. Eng.*, 30 :587–600, September 2004.

[Mak77] Gennadi Semyonovich Makanin. The problem of solvability of equations in a free semi-group. *Math. USSR Sbornik*, 32(2) :129–198, 1977.

[Mat89] Friedemann Mattern. Virtual Time and Global State in Distributed Systems. In *Int'l Workshop on Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers, 1989.

[MAT10] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[MD96] Subrata Mitra and Nachum Dershowitz. Matching and Unification in Rewrite Theories, 1996.

[Mil82] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.

[Mil99] Robin Milner. *Communicating and Mobile Systems : The π-calculus*. Cambridge University Press, 1999.

[MJ00] B. Meyer and P. Jouvelot. *Conception et programmation orientées objet*. Technologies objet (Eyrolles, Paris). Eyrolles, 2000.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2) :258–282, 1982.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., 1992.

[MR92] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 195–202, New York, NY, USA, 1992. ACM.

[MvH04] Deborah McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview, W3C Recommendation, 2004.

[MW93] I. Scott MacKenzie and Colin Ware. Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 488–493, New York, NY, USA, 1993. ACM.

[NSD+01] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubezy, Ray W. Fergerson, and Mark A. Musen. Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, 2(16) :60–71, 2001.

[Pee05] J. Peer. *Web Service Composition as AI Planning : A Survey*. University of St. Gallen, 2005.

[PS11] Andriy Pavlovych and Wolfgang Stuerzlinger. Target following performance in the presence of latency, jitter, and signal dropouts. In *Proceedings of Graphics Interface 2011*, GI '11, pages 33–40. Canadian Human-Computer Communications Society, 2011.

[PW76] Mike Paterson and Mark Wegman. Linear unification. In *STOC '76 : Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 181–186, New York, NY, USA, 1976. ACM Press.

[Rit89] Mikael Rittri. Using types as search keys in function libraries. In *FPCA '89 : Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 174–183, New York, NY, USA, 1989. ACM Press.

[RLS98] R. Raman, M. Livny, and M. H. Solomon. Matchmaking : Distributed resource management for high throughput computing. In *HPDC*, pages 140–, 1998.

[RT89] Colin Runciman and Ian Toyn. Retrieving reusable software components by polymorphic type. In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, pages 166–173, London, UK, 1989. ACM Press.

[SCG+00] Mihaela Sighireanu, Claude Chaudet, Hubert Garavel, Marc Herbert, Radu Mateescu, and Bruno Vivien. LOTOS NT user manual, 2000.

[Sch92] Klaus Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *IWWERT '90 : Proceedings of the First International Workshop on Word Equations and Related Topics*, pages 85–150, London, UK, 1992. Springer-Verlag.

[Sci12] Scilab Enterprises. *Scilab : Le logiciel open source gratuit de calcul numérique*. Scilab Enterprises, Orsay, France, 2012.

[SHQ17] Adam Shimi, Aurélie Hurault, and Philippe Quéinnec. Inference of Channel Priorities for Asynchronous Communication. In *21th International Conference on Principles of Distributed Systems, OPODIS 2017, December 18-20, 2017, Lisboa, Portugal*, 2017.

[SHQ18] Nathanaël Sensfelder, Aurélie Hurault, and Philippe Quéinnec. Inference of Channel Priorities for Asynchronous Communication. In *Distributed Computing and Artificial Intelligence, 14th International Conference*, pages 262–269, Cham, 2018. Springer International Publishing.

[Sie79] Jörg Siekmann. Unification of commutative terms. In *EUROSAM '79 : Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, page 22, London, UK, 1979. Springer-Verlag.

[Spi89] J. M. Spivey. *The Z Notation : A Reference Manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[SS96] Manfred Schmidt-Schauß. Decidability of unification in the theory of one-sided distributivity and a multiplicative unit. *J. Symb. Comput.*, 22(3) :315–344, 1996.

[TM96] Gadi Taubenfeld and Shlomo Moran. Possibility and impossibility results in a shared memory environment. *Acta Informatica*, 33(1) :1–20, Feb 1996.

[TNS+03] Yoshio Tanaka, Hidemoto Nakada, Satoshi Sekiguchi, Toyotaro Suzumura, and Satoshi Matsuoka. Ninf-G : A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1) :41–51, 2003.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice : the Condor experience. *Concurrency - Practice and Experience*, 17(2-4) :323–356, 2005.

[vdm78] *The Vienna Development Method : The Meta-Language*, London, UK, UK, 1978. Springer-Verlag.

[WB94] Colin Ware and Ravin Balakrishnan. Reaching for objects in vr displays : Lag and frame rate. *ACM Trans. Comput.-Hum. Interact.*, 1(4) :331–356, December 1994.

[WPD01] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1–2) :3–35, 2001.

[YB12] Qi Yu and Athman Bouguettaya. Multi-attribute optimization in service selection. *World Wide Web*, 15(1) :1–31, 2012.

[YS97] Daniel M. Yellin and Robert E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2) :292–333, March 1997.

[ZBN+04] Liangzhao Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5) :311–327, May 2004.

[ZW93] Amy Moormann Zaremski and Jeannette M. Wing. Signature Matching : A Key to Reuse. In *Proceedings of the first ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 182–190. ACM Press, 1993.