HAL
open science

Intergiciel d'intergiciels adaptable à base de Services, Composants et Aspects

Philippe Merle

To cite this version:

Philippe Merle. Intergiciel d'intergiciels adaptable à base de Services, Composants et Aspects. Calcul parallèle, distribué et partagé [cs.DC]. Université Lille 1 Sciences et Technologies, 2015. tel-01206244

HAL Id: tel-01206244

https://theses.hal.science/tel-01206244v1

Submitted on 28 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

HABILITATION À DIRIGER DES RECHERCHES DE L'UNIVERSITÉ LILLE 1, SCIENCES ET TECHNOLOGIES

Spécialité

Informatique

École doctorale Sciences Pour l'Ingénieur (Lille)

Présentée par

Philippe MERLE

Titre de l'habilitation :

Intergiciel d'intergiciels adaptable à base de Services, Composants et Aspects

soutenue le 24 septembre 2015

devant le jury composé de :

DR	Frédéric Desprez	Rapporteur
Pr.	Rachid Guerraoui	Rapporteur
Pr.	Jean-Marc Jézéquel	Rapporteur
DR	Jean-Bernard Stefani	Examinateur
Pr.	Jean-Marc Geib	Examinateur
Pr.	Lionel Seinturier	Garant

Remerciements

Je remercie très chaleureusement les professeurs Rachid Guerraoui, Jean-Marc Jézéquel et le directeur de recherche Frédéric Desprez pour avoir accepté d'être rapporteurs de mon habilitation à diriger des recherches, ainsi que le professeur Jean-Marc Geib et le directeur de recherche Jean-Bernard Stefani pour avoir accepté de participer à mon jury. Je remercie très vivement le professeur Lionel Seinturier d'avoir accepté d'être le garant de cette habilitation et de m'avoir conseillé durant la rédaction et préparation de celle-ci.

Je tiens à remercier fortement tous les collègues avec qui j'ai collaboré sur les deux dernières décennies. Merci aux professeurs Jean-Marc Geib, Laurence Duchien et Lionel Seinturier pour m'avoir accueilli respectivement dans leur équipe GOAL puis Jacquard, ADAM et Spirals. Un grand merci à tous les doctorants, étudiants et ingénieurs que j'ai (co-)encadré : Raphaël Marvie, Sylvain Leblanc, Mathieu Vadet, Romain Rouvoy, Christophe Contreras, Christophe Demarey, Patricia Serrano-Alvarado, Areski Flissi, Frédéric Briclet, Jérémy Dubus, Aleš Plšek, Yann Davin, Alban Tiberghien, Damien Fournier, Nicolas Dolet, Virginie Legrand Contes, Jonathan Labéjof, Christophe Munilla, Antonio de Almeida Souza Neto, Michel Dirix, Gwenaël Cattez, Mathieu Schepens et Fawaz Paraiso. Un grand merci aussi à tous mes autres collègues et co-auteurs. Sans vous tous, je n'aurais pas pu mener à bien les travaux présentés dans ce manuscrit.

Merci à mon épouse Razia et nos deux filles, Thaïs et Maëlys, pour l'amour, le bonheur et le soutien que vous m'apportez au quotidien. Enfin, j'ai une pensée émue pour ma défunte épouse Valérie qui s'est envolée trop tôt.

Table des matières

Remerciements **iii**

1 Introduction **1**
1.1 Parcours professionnel . . . 1
1.2 Domaine de recherche . . . 2
1.3 Contributions . . . 3
1.3.1 De CorbaScript à OMG IDLscript (1997 - 2003) . . . 4
1.3.2 Des objets aux composants CORBA (1997 - 2008) . . . 4
1.3.3 Canevas intergiciels hautement adaptables à base de composants réflexifs Fractal (2003 - 2009) . . . 7
1.3.4 Le modèle Services Composants Aspects de l'intergiciel d'intergiciels FraSCAti (2007 - 2015) . . . 9
1.4 Organisation du manuscrit . . . 11

2 Adaptabilité de services distribués **13**
2.1 Introduction historique . . . 14
2.2 Un rapide survol du modèle SCA . . . 15
2.2.1 La genèse de SCA . . . 15
2.2.2 Le modèle d'assemblage SCA . . . 16
2.2.3 L'application hétérogène Twitter - Weather . . . 18
2.3 Questions de recherche . . . 19
2.4 La genèse de FraSCAti . . . 21
2.5 QR1 : Supporter la flexibilité / extensibilité / adaptabilité du modèle SCA . . . 23
2.5.1 Des composants partout . . . 23
2.5.2 Un intergiciel d'intergiciels . . . 24
2.5.3 L'intégration de multiples implantations . . . 26
2.5.3.1 Le cas de l'intégration du langage WS-BPEL . . . 26
2.5.4 L'interopérabilité de multiples liaisons . . . 28
2.5.4.1 Les composants de liaison . . . 29
2.5.5 Les besoins extra-fonctionnels . . . 30
2.5.5.1 Les composants d'aspect . . . 30
2.5.5.2 La capture de diagramme de séquence UML . . . 32
2.5.5.3 Les aspects de liaisons Apache CXF . . . 33
2.5.5.4 Model Driven Security@run.time . . . 35
2.5.6 La plate-forme FraSCAti . . . 38
2.5.7 Synthèse sur QR1 . . . 40

2.6 QR2 : Modulariser la flexibilité / extensibilité / adaptabilité d'une plate-forme SCA . . . 41
2.6.1 Une plate-forme modulable " à la carte " . . . 42
2.6.2 La fragmentation de la description de l'architecture de la plate-forme . . 43
2.6.3 La recomposition de l'architecture de la plate-forme FraSCAti . . . 44
2.6.4 Synthèse et perspectives sur QR2 . . . 45
2.7 QR3 : Reconfigurer à l'exécution des applications SCA . . . 46
2.7.1 Equiper les composants de capacités réflexives . . . 46
2.7.2 L'interface réflexive de FraSCAti . . . 47
2.7.3 L'outil FraSCAti Explorer . . . 49
2.7.4 L'outil FraSCAti JMX . . . 49
2.7.5 L'interface FraSCAti REST . . . 50
2.7.6 La console FraSCAti Web Explorer . . . 50
2.7.7 Le langage FraSCAti FScript . . . 50
2.7.8 Bilan et perspectives sur QR3 . . . 53
2.8 QR4 : Modéliser la variabilité de la plate-forme FraSCAti . . . 54
2.8.1 La modélisation des fonctionnalités de la plate-forme FraSCAti . . . 54
2.8.2 L'extraction automatique du modèle de fonctionnalités de la plate-forme FraSCAti . . . 57
2.8.3 L'évolution du modèle de fonctionnalités de la plate-forme FraSCAti . 61
2.9 Synthèse sur FraSCAti . . . 62
2.10 L'orchestration décentralisée et dynamique de processus métiers à base de services 66
2.10.1 Le contexte . . . 66
2.10.2 La problématique . . . 67
2.10.3 Les contributions . . . 68
2.11 La réflexivité au service de l'évolution des systèmes de systèmes . . . 71
2.11.1 Le contexte et la problématique . . . 71
2.11.2 Les contributions . . . 72
2.11.2.1 R-DDS : la reconfigurabilité de l'intergiciel DDS . . . 72
2.11.2.2 R-MOM : l'interopérabilité des intergiciels asynchrones . . . 75
2.11.2.3 R-EMS : la spécification de systèmes de systèmes . . . 76
2.12 La plate-forme multi-nuages soCloud . . . 77
2.12.1 Le contexte et la problématique . . . 77
2.12.2 Les contributions . . . 79
2.12.2.1 Le modèle soCloud . . . 79
2.12.2.2 La plate-forme soCloud . . . 81
2.12.2.3 Trois applications soCloud . . . 82

3 Bilan et perspectives 87
3.1 Bilan . . . 87
3.2 Perspectives . . . 88

Bibliographie 92

Table des figures

1.1 Deux questions de recherche ouvertes sur les intergiciels. 3

2.1 Un composite SCA. 16
2.2 L'application Twitter - Weather. 18
2.3 Quatre questions de recherche pour l'adaptabilité de services distribués. 20
2.4 L'organisation du canevas SOFA. Source : Francis Palma. 23
2.5 La couche intergicielle intermédiaire entre les applications et les systèmes. Source : [13, 76]. 24
2.6 FraSCAti : un intergiciel d'intergiciels. 25
2.7 L'intégration d'EasyBPEL avec FraSCAti. 27
2.8 Les composants de liaison d'interopérabilité. 29
2.9 Les composants extra-fonctionnels. Source : [214]. 30
2.10 La structure minimale d'un composant d'aspect FraSCAti. 31
2.11 L'aspect FraSCAti de capture de diagrammes de séquences UML. 32
2.12 Un exemple de capture de diagramme de séquence UML. 33
2.13 Un scénario de répartition de la charge sur un cluster de services Web. 34
2.14 La représentation graphique du composite du listing 2.3. 36
2.15 L'architecture de MDS@run.time. Source : [163]. 37
2.16 La plate-forme FraSCAti. 38
2.17 L'architecture simplifiée de l'interpréteur FraSCAti de descripteurs SCA. . . . 39
2.18 Les modules de FraSCAti et le graphe de leurs dépendances. 42
2.19 La fragmentation du composite `Assembly Factory`. 44
2.20 La recomposition du composite `Assembly Factory`. 45
2.21 La pile réflexive de la plate-forme FraSCAti. 47
2.22 L'assemblage des composants de contrôle d'une membrane de composants FraSCAti. 48
2.23 Comparaison de FraSCAti et Apache Tuscany. Source : [214]. 49
2.24 FraSCAti Explorer : un microscope pour introspecter et reconfigurer des applications SCA. 50
2.25 FraSCAti JMX. 51
2.26 L'interface de programmation FraSCAti REST. 51
2.27 FraSCAti Web Explorer. 52
2.28 Une reconfiguration avec FraSCAti FScript. 52
2.29 Les points de variabilité de l'architecture de FraSCAti. 55
2.30 Un extrait du diagramme de fonctionnalités de FraSCAti. Source : [2]. 56
2.31 Le diagramme de fonctionnalités de FraSCAti 1.4. 57

2.32 Le processus d'extraction du modèle de fonctionnalités de FraSCAti. Source : [3]. 58
2.33 L'extraction des fonctionnalités depuis l'architecture de FraSCAti. Source : [3]. 59
2.34 Le modèle de fonctionnalités de FraSCAti publié sur le site S.P.L.O.T.. . . . 60
2.35 L'évolution du modèle de fonctionnalités de FraSCAti. Source : [3]. 61
2.36 Un intergiciel d'intergiciels. 62
2.37 Des composants à tous les étages. 63
2.38 Le lien causal entre le modèle de fonctionnalités et l'architecture modulaire de FraSCAti. 64
2.39 L'architecture du projet SOA4All. Source : [10]. 66
2.40 Un processus métier de réservation d'un voyage. Source : [87]. 67
2.41 Le découpage spatio-temporelle du processus métier de réservation de voyages. Source : [87]. 68
2.42 Les fragments composant un processus métier. Source : [87]. 69
2.43 La composition de fragments. Source : [87]. 70
2.44 Le système de systèmes TACTICOS. Source : [80]. 72
2.45 L'approche R-* comprenant les contributions R-DDS, R-MOM et R-EMS. . . . 73
2.46 Le patron de construction des composants R-DDS. 73
2.47 Comparaison des temps moyens de traitement entre DDS, R-DDS et R-MOM DDS. Source : [80]. 74
2.48 Le diagramme des composants R-MOM. Source : [78, 80]. 75
2.49 Une passerelle R-MOM d'interopérabilité de DDS vers JMS. 76
2.50 L'informatique en nuage. Source : Wikipédia. 78
2.51 Le modèle et la plate-forme soCloud. 79
2.52 Une application soCloud. Source : [174]. 80
2.53 Les annotations soCloud. Source : [174]. 80
2.54 L'architecture à composants de la plate-forme soCloud. Source : [174]. 81
2.55 Le déploiement de soCloud dans les nuages. Source : [174]. 83
2.56 La plate-forme APISENSE. Source : [174]. 84
2.57 La plate-forme DiCEPE. Source : [174]. 85
2.58 L'application de monitoring pair à pair dans les nuages. Source : [134]. 86

3.1 Vers un intergiciel pour le Cloud Computing. 89

Liste des tableaux

2.1 Les temps moyens d'exécution des composantes de MDS@RUN.TIME. 37

Listings

2.1 La description SCA simplifiée de l'application Twitter - Weather 18
2.2 L'implantation minimale en Java d'un composant d'aspect 31
2.3 Le composite de l'application cliente d'un cluster de services Web 35
2.4 L'interface Java d'un processeur FraSCAti 39
2.5 Extrait du composite `Assembly Factory` du module `FraSCAti Core` 43
2.6 Le composite `Assembly Factory` du module `Interface WSDL` 43
2.7 Le composite `Assembly Factory` du module `Implementation BPEL` 44
2.8 Un script de reconfiguration en FraSCAti FScript 53

Chapitre 1

Introduction

Sommaire

1.1 Parcours professionnel **1**
1.2 Domaine de recherche **2**
1.3 Contributions **3**
1.3.1 De CorbaScript à OMG IDLscript (1997 - 2003) 4
1.3.2 Des objets aux composants CORBA (1997 - 2008) 4
1.3.3 Canevas intergiciels hautement adaptables à base de composants réflexifs Fractal (2003 - 2009) 7
1.3.4 Le modèle Services Composants Aspects de l'intergiciel d'intergiciels FraSCAti (2007 - 2015) 9
1.4 Organisation du manuscrit **11**

Ce chapitre introductif retrace les grandes lignes de mon parcours professionnel (*cf.* Section 1.1), introduit le domaine de recherche et les deux principales questions de recherche auxquels je me suis attelés (*cf.* Section 1.2), résume mes principales contributions (*cf.* Section 1.3) et présente l'organisation de ce manuscrit (*cf.* Section 1.4).

1.1 Parcours professionnel

En 1992, j'ai intégré l'équipe Groupe Objets et Acteurs de Lille (GOAL) du Laboratoire d'Informatique de Lille (LIFL) de l'Université des Sciences et Technologies de Lille (USTL) dirigée par le Prof. Jean-Marc Geib afin d'effectuer mon stage de recherche en DEA sur le langage de programmation répartie BOX [60, 53, 54, 55]. Après dix mois passés sous les drapeaux, j'ai démarré en octobre 1993 mon doctorat en informatique sous la direction du Prof. Jean-Marc Geib au sein de l'équipe GOAL sur le thème des objets répartis et plus spécifiquement autour de la technologie émergente nommée CORBA. Après un peu plus de trois années, j'ai obtenu en janvier 1997 mon doctorat intitulé " CorbaScript - CorbaWeb : propositions pour l'accès à des objets et services répartis " [114]. En septembre 1997, j'ai été recruté comme Maître de Conférences à l'USTL où j'ai poursuivi mes activités de recherche au sein de l'équipe GOAL. En septembre 2001, j'ai obtenu une année de délégation chez INRIA afin de travailler sur le programme de recherche sur l'intergiciel OpenCCM pour les composants répartis CORBA. En septembre 2002, j'ai été recruté comme Chargé de Recherche INRIA avec

comme perspective la création d'une nouvelle équipe-projet INRIA nommée Jacquard (2003 - 06). Cette équipe sous la direction du Prof. Jean-Marc Geib avait pour thème de recherche le tissage de composants logiciels et la programmation orientée aspect (AOP). Puis, Jacquard a été suivi par l'équipe-projet ADAM [1] (2007 - 13) sous la direction du Prof. Laurence Duchien et ayant pour thème de recherche l'adaptation des intergiciels et des applications réparties. Au sein de cette équipe, j'ai initié le programme de recherche sur l'intergiciel FraSCAti. Depuis 2014, je suis membre de l'équipe-projet Spirals [2] dirigée par le Prof. Lionel Seinturier, ayant comme thème de recherche les systèmes auto-adaptatifs et étudiant plus particulièrement les propriétés d'auto-optimisation et d'auto-réparation.

Ainsi en regardant dans le rétroviseur, je m'aperçois que j'ai déjà vécu plus de vingt années au sein de quatre équipes de recherche. Il est vraiment temps de faire un bilan avant d'attaquer les vingt prochaines années qui seront aussi voire encore plus passionnantes et enrichissantes. Ce manuscrit retrace mes travaux de recherche (*cf.* Section 1.3) depuis l'obtention de mon doctorat en 1997 et se concentre principalement sur les travaux menés depuis 2007 autour de l'**intergiciel d'intergiciels FraSCAti** (*cf.* Chapitre 2).

1.2 Domaine de recherche

Mes travaux de recherche s'inscrivent à l'intersection de deux domaines de recherche : l'informatique distribuée et le génie logiciel. Mon domaine de recherche peut ainsi être vu comme **le génie logiciel pour l'informatique distribuée** et plus précisément **le génie logiciel des intergiciels**. Ce domaine porte sur la conception et la réalisation de modèles et de plates-formes logiciels (*cf.* génie logiciel) pour la construction, le déploiement, l'hébergement, l'exécution et l'administration d'applications réparties (*cf.* informatique distribuée).

Les applications réparties sont confrontées en autres aux problèmes suivants :

- l'**hétérogénéité** des technologies sous-jacentes aussi bien au niveau des réseaux de télécommunications, des composantes matérielles des machines, des systèmes d'exploitation et des langages de programmation des applications,
- l'**interopérabilité** entre applications s'exécutant sur des technologies hétérogènes,
- la **portabilité** des applications dans des environnements hétérogènes,
- l'**adaptabilité** des applications à des contextes d'exécution variables et fluctuants, et
- la **séparation des préoccupations** techniques telles que la désignation, les interactions, la sécurité, les transactions, la tolérance aux pannes, etc.

Afin d'adresser ces problèmes, il est nécessaire de mettre en place une couche intermédiaire entre les applications et les systèmes sous-jacents (systèmes d'exploitation, matériels et réseaux) comme l'illustre la figure 1.1. Cette couche est appelée **intergiciel** ou *middleware* en anglais. Comme énoncé dans [13, 76], l'intergiciel masque la distribution des applications et l'hétérogénéité des systèmes sous-jacents, tout en fournissant des abstractions et des interfaces de programmation de haut niveau pour structurer et développer les applications ainsi que des fonctions techniques de base telles que interaction répartie, annuaire, transaction, sécurité, tolérance aux pannes, etc.

Dans ce contexte, je me suis intéressé plus particulièrement à deux questions de recherche ouvertes [3] :

1. `http://adam.lille.inria.fr`
2. `https://team.inria.fr/spirals/`
3. Ces questions sont ouvertes car elles n'ont pas une réponse unique et consensuelle.

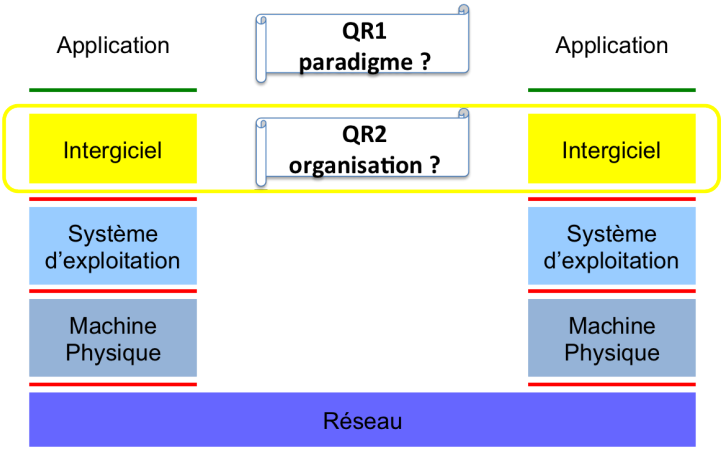

FIGURE 1.1 – Deux questions de recherche ouvertes sur les intergiciels.

- **QR1 - Quel est le paradigme le plus approprié pour les applications ?** Dans la littérature, il existe de nombreux paradigmes pour organiser, structurer et développer des applications réparties. Ces paradigmes sont mis en oeuvre par l'intergiciel. Par ordre chronologique, on peut citer en autres les paradigmes à base de procédures, d'acteurs, d'objets, de composants, de services et de ressources répartis. Au cours de mes travaux de recherche, j'ai principalement contribué autour des objets répartis CORBA, des composants CORBA puis FRACTAL puis SCA, des services et ressources Web.
- **QR2 - Quelle est l'organisation la plus appropriée pour l'intergiciel ?** L'intergiciel est un logiciel complexe nécessitant une organisation structurée au même titre que les applications métiers. Au cours de mes travaux de recherche, je me suis particulièrement intéressé à l'utilisation de la programmation orientée composant pour organiser l'intergiciel afin que celui-ci soit portable et adaptable à différents contextes d'exécution.

Pour cela, j'ai cherché à proposer un modèle commun pour construire / déployer / héberger / administrer aussi bien les applications métiers que la couche intergiciel. L'idée sous-jacente fut de casser la frontière imperméable entre les applications et l'intergiciel, c'est-à-dire que les applications participent à la réalisation des fonctionnalités de l'intergiciel et que l'intergiciel est vu au même titre que les applications métiers. Cette quête du Graal est présentée dans la section suivante.

1.3 Contributions

Cette section résume mes principales contributions depuis l'obtention de mon doctorat en 1997. J'ai réparti ces contributions en quatre grandes périodes :

1. de CORBASCRIPT à OMG IDLSCRIPT (1997 - 2003) (*cf.* Section 1.3.1),
2. des objets aux composants CORBA (1997 - 2008) (*cf.* Section 1.3.2),
3. canevas intergiciels hautement adaptables à base de composants réflexifs FRACTAL (2003 - 2009) (*cf.* Section 1.3.3) et
4. le modèle Services Composants Aspects de l'intergiciel d'intergiciels FRASCATI (2007 - 2015) (*cf.* Section 1.3.4).

1.3.1 De CorbaScript à OMG IDLscript (1997 - 2003)

Dans ma thèse de doctorat [114], j'ai proposé deux outils pour faciliter l'accès à des objets et services répartis [128, 123, 113]. CorbaWeb est une passerelle Web pour naviguer, visualiser et invoquer des objets CORBA [126, 127]. CorbaWeb définit une projection des concepts de CORBA vers les concepts du Web : objet CORBA vers représentation HTML, référence CORBA vers hyper-lien, invocation CORBA vers formulaire HTML, etc. L'association de cette passerelle et d'un navigateur Web fournit alors un navigateur générique sur tout objet CORBA [125, 121]. Cette passerelle fut mise en oeuvre sur l'intégration de trois environnements hétérogènes, à savoir le World Wide Web, la carte à microprocesseur et CORBA [137, 138]. La passerelle CorbaWeb est construite au dessus du langage de scripts CorbaScript [124]. Ce langage orienté objet et à typage dynamique permet d'invoquer tout objet CORBA via les mécanismes réflexifs de CORBA, c'est-à-dire le référentiel des interfaces (*cf. Interface Repository*), les interfaces d'invocation et d'implantation dynamique (*cf. Dynamic Invocation Interface (DII)* et *Dynamic Skeleton Interface (DSI)*).

Cette première période fut consacrée à la promotion de CorbaScript puis à sa standardisation auprès du consortium international Object Management Group (OMG).

Promotion de CorbaScript Nous avons présenté comment concevoir, déployer et utiliser des services distribués avec CorbaScript dans [129]. Nous avons discuté l'utilisation et l'implantation d'objets CORBA avec CorbaScript dans [132, 133]. Nous avons introduit une nouvelle approche pour construire des outils génériques pour exploiter CORBA dans [130, 131]. Nous avons collaboré avec Serge Du et Oleg Lodygensky de l'institut national de physique nucléaire et de physique des particules (IN2P3) pour utiliser CorbaScript dans le domaine de la physique des hautes énergies [122, 119, 120]. Nous avons mené les premières expériences d'utilisation des composants CORBA avec CorbaScript dans [101, 102].

Standard OMG IDLscript A partir de 1997, nous avons répondu à un appel à propositions initié par l'OMG pour la standardisation d'un langage de scripts pour CORBA. Après un long processus de standardisation, notre proposition basée sur le langage CorbaScript est devenu le standard OMG *CORBA Scripting Language* nommé IDLscript en 2001 [115] puis révisé en 2003 [116].

CorbaScript et IDLscript sont deux contributions à ma première question de recherche (QR1). Ils offrent un paradigme facilitant la construction d'applications CORBA.

1.3.2 Des objets aux composants CORBA (1997 - 2008)

Cette deuxième période regroupe mes travaux sur la transition des objets vers les composants CORBA.

Ouvrage de référence sur CORBA Suite à l'expertise sur CORBA acquise durant ma thèse de doctorat, j'ai co-écrit un ouvrage de référence sur CORBA intitulé " CORBA : des concepts à la pratique " édité chez Masson en 1997 [56] puis réédité chez Dunod en 1999 [57]. Une version condensée fut publiée dans la revue Techniques de l'Ingénieur, traité Informatique en 2000 [58].

Cet ouvrage est une contribution à ma première question de recherche (QR1). CORBA était un paradigme facilitant la construction d'applications réparties.

Standardisation à l'OMG autour des composants CORBA De 2001 à 2003, j'ai été le *chair* des groupes de finalisation puis de révision du standard OMG sur les composants CORBA. Ma contribution a principalement porté sur l'animation et le pilotage de ces groupes de travail OMG, la correction de la spécification des composants CORBA et la rédaction de nombreux documents OMG intermédiaires. Techniquement, j'ai complété les capacités d'introspection du standard " *CORBA Components* ". J'ai été le co-éditeur de la spécification OMG " *CORBA Components* " [117]. Puis, j'ai participé à la spécification OMG d'un profil UML pour les composants CORBA [160]. Durant cette période, j'ai disséminé le modèle de composants CORBA au sein de la communauté française [195, 91, 96].

Cet effort de standardisation est une contribution à ma première question de recherche (QR1). Les composants CORBA furent un paradigme facilitant la construction d'applications réparties.

Séparation des préoccupations dans les architectures logicielles à base de composants CORBA De 1998 à 2002, j'ai co-encadré avec le Prof. Jean-Marc Geib la thèse de Raphaël Marvie portant sur la séparation des préoccupations et méta-modèlisation pour environnements de manipulation d'architectures logicielles à base de composants [93]. Cette thèse propose un cadre de travail nommé CODEX [94] pour méta-modéliser des langages de description d'architectures (ADL) [108]. L'intérêt principal de ce cadre est de promouvoir la séparation des préoccupations au sein d'un ADL [100]. Ainsi chaque préoccupation d'un ADL est traitée dans un méta-modèle indépendant puis l'ensemble des préoccupations sont composées pour former l'ADL final. Cela permet ainsi de construire des ADLs à la carte. Ce cadre a été mis en oeuvre pour construire un langage de description d'architectures à base de composants CORBA. Ce travail fut poursuivi dans la thèse de Bassem Kosayba afin de construire des environnements graphiques dédiés à un domaine [72, 73, 74].

Cette thèse fut une contribution à ma première question de recherche (QR1). CODEX est un cadre facilitant la construction de langages de description d'architectures pour applications réparties.

Conteneurs ouverts pour composants CORBA En 2001, j'ai encadré le mémoire de DEA de Mathieu Vadet sur les conteneurs ouverts dans les plates-formes à composants [224, 226] puis de 2002 à 2004, j'ai co-encadré avec le Prof. Jean-Marc Geib la thèse de Mathieu Vadet sur un modèle de services logiciels pour la spécialisation des intergiciels à composants [225]. Cette thèse CIFRE a été financée par la société Thalès et effectuée dans le cadre du projet européen IST COACH qui avait pour objectif de construire une plate-forme à composants CORBA pour le domaine des télécommunications [112, 15]. L'objectif de ce travail fut de concevoir un modèle ouvert pour la construction de conteneurs de composants CORBA. Ce modèle permet de séparer les préoccupations prises en charge par un conteneur tels que le cycle de vie des composants, la sécurité, les transactions, les connecteurs entre composants [227], etc. Chacune de ces préoccupations est réalisée sous la forme d'un service logiciel indépendant puis l'ensemble des préoccupations sont composées pour former un conteneur à la carte.

Cette thèse fut une contribution à ma deuxième question de recherche (QR2) en proposant un modèle ouvert et extensible pour organiser les conteneurs de composants CORBA.

Composants de courtage CORBA De 2000 à 2003, nous avons travaillé sur la structuration d'un service de courtage d'objets et de composants répartis CORBA. La fonction de courtage d'un intergiciel permet de rechercher des services répondant à un ensemble de critères aussi bien statiques que dynamiques. Dans le cadre du mémoire de DEA de Sylvain Leblanc que j'ai encadré [81], nous avons proposé un langage dédié à l'expression de contrats de courtage [103, 104, 105, 82, 85]. Ce langage permet de générer des proxies fortement typés masquant les interactions de bas niveau avec le courtier [106]. Ces proxies de courtage sont ensuite encapsulés dans des composants CORBA afin de faciliter leur déploiement et leur partage [83, 84]. Ces composants de courtage furent mis en oeuvre dans le projet RNTL COMPiTV [24] visant une plate-forme pour la télévision interactive à base de composants. Malheureusement, Sylvain Leblanc ne put jamais soutenir la thèse qu'il avait démarré car il est décédé prématurément !

Cet travail fut une contribution à ma deuxième question de recherche (QR2) en proposant un modèle ouvert et extensible pour structurer la fonction de courtage d'un intergiciel.

Déploiement réparti de composants CORBA De 2003 à 2008, nous nous sommes attaqués au déploiement d'applications réparties à base de composants CORBA. Dans le cadre du projet IST COACH, j'ai participé en 2003 à la rédaction d'une spécification pour le déploiement et la configuration de composants CORBA [67]. Ce travail fut la base de la spécification OMG " *Deployment and Configuration of Component-based Distributed Applications* " (OMG D&C). En 2004, j'ai encadré le mémoire de DEA de Frédéric Briclet [17] qui a proposé une implantation de cette spécification. L'originalité de cette infrastructure de déploiement distribuée est qu'elle est elle-même réalisée en composants CORBA [18]. Ainsi cette infrastructure est auto-déployable. En 2005 dans le cadre du projet régional MOSAIQUES, nous avons spécialisé cette infrastructure pour pouvoir l'exécuter sur des *smart phones* et déployer ainsi des applications ubiquitaires [49, 48, 47] et plus particulièrement dans le domaine des transports [46]. De 2005 à 2008, j'ai encadré le mémoire de DEA de Jérémy Dubus sur la modélisation et la génération automatique d'infrastructures de déploiement d'applications réparties [33] puis j'ai co-encadré avec le Prof. Jean-Marc Geib la thèse de Jérémy Dubus portant sur une démarche orienté modèle pour le déploiement de systèmes en environnements ouverts distribués [34]. Nous avons travaillé sur l'auto-adaptabilité des architectures logicielles dans les environnements ouverts distribués [37, 36] en se basant sur la spécification OMG D&C et des règles ECA [35]. Nous avons proposé une technique de validation dirigée par les modèles pour les systèmes logiciels autonomes en environnements ouverts distribués [38]. Enfin, nous avons proposé une démarche orientée modèle pour déployer des systèmes logiciels répartis [39].

Ces travaux contribuèrent à ma deuxième question de recherche (QR2) en proposant une infrastructure pour le déploiement d'applications réparties à base de composants CORBA.

Logiciel OpenCCM L'ensemble de ces travaux de recherche a été intégré dans le logiciel OpenCCM [4]. Ce logiciel en licence libre LGPL est hébergé par le consortium international OW2 dédié aux intergiciels depuis 2002. OpenCCM est une implantation de référence du standard OMG CORBA Components. Ce logiciel contient plus de 444 500 lignes de code

4. `http://openccm.ow2.org`

principalement en Java. Je suis le responsable de ce projet OW2, son principal architecte et son principal développeur avec environ 22% des commits. La conception et l'implantation du logiciel OpenCCM ont été discutées dans [92, 98, 97, 99, 95, 107].

Ce logiciel OpenCCM est une contribution à ma deuxième question de recherche (QR2) en proposant une plate-forme intergicielle ouverte pour le déploiement, l'hébergement et l'exécution de composants logiciels conformes au standard *CORBA Components*.

1.3.3 Canevas intergiciels hautement adaptables à base de composants réflexifs Fractal (2003 - 2009)

Cette troisième période regroupe mes travaux sur la construction de canevas intergiciels hautement adaptables à base de composants réflexifs Fractal. Ces travaux ont été initiés dans la thèse de Romain Rouvoy proposant une démarche à granularité extrêmement fine pour la construction de canevas intergiciels hautement adaptables avec une application aux services de transactions [199], que j'ai co-encadré avec le Prof. Jean-Marc Geib.

Canevas pour le transactionnel En 2003, j'ai encadré le mémoire de DEA de Romain Rouvoy sur la conception du canevas GoTM pour la gestion des aspects transactionnels dans les plates-formes à composants [198] puis publié dans [201]. La partie de ce canevas dédiée à la gestion de la démarcation des transactions dans les plates-formes à composants a été publiée à la conférence Middleware 2003 [200]. Dans [215], nous avons montré comment rendre la gestion du protocole de validation des transactions auto-adaptable, c'est-à-dire que le gestionnaire de transactions choisit le protocole de validation (normal, optimiste, pessimiste) en fonction des réussites ou échecs des transactions précédentes. Dans [211], nous avons montré comment un service de transactions pourrait être conscient du contexte dans lequel il s'exécute. Dans [210], nous avons montré comment composer des standards de gestion de transactions hétérogènes (OTS, JTS, WS-AT) à l'aide de notre canevas. Dans [203] puis [205], nous avons montré comment construire des services de transactions évolutionnistes en utilisant des micro-composants et des patrons de conception. Pour cela, nous avons étendu le langage Fractal ADL pour factoriser les motifs se répétant d'une architecture à une autre [204]. L'ensemble de ces travaux est présenté dans la thèse de Romain Rouvoy [199].

Ces travaux contribuent à ma deuxième question de recherche (QR2) en proposant une organisation à base de composants de la gestion des transactions dans un intergiciel.

Programmation par annotations En 2006, Romain et moi avons proposé de faciliter le développement des composants logiciels en utilisant la programmation par annotations [208]. Cela consiste à annoter le code métier des composants avec des méta-données utilisés par le canevas d'exécution de ces composants. Nous avons montré que cela permet de réduire fortement la quantité de code technique que le programmeur doit développer classiquement. Dans [209], nous avons proposé un jeu d'annotations pour le modèle de composants Fractal. Dans [202], nous avons montré que ce même jeu d'annotations pourrait aussi être utilisé pour le développement de composants conformes au modèle OpenCom. Les composants sont ainsi portables sur plusieurs canevas d'exécution. Ce jeu d'annotations nommé Fraclet a été intégré au projet OW2 Fractal et a été depuis largement adopté et utilisé par la communauté Fractal. Depuis, la programmation par annotations est devenue la norme dans tous les modèles de programmation orientée composants. La synthèse sur ce travail a été publié en 2009 dans la revue *Annals of Telecommunications* [206].

Ce travail contribue à ma première question de recherche (QR1) en facilitant le développement de grandes bibliothèques de composants logiciels.

Canevas de déploiement de systèmes distribués hétérogènes De 2006 à 2008, nous avons développé un canevas pour faciliter le déploiement de systèmes logiciels distribués et hétérogènes [50]. Ce canevas à base de composants FRACTAL se nomme FRACTAL DEPLOYMENT FRAMEWORK (FDF) [5]. FDF supporte tout type d'activités de déploiement tels que le téléchargement, l'installation, la configuration, le démarrage, l'arrêt et la désinstallation de tout type de piles logiciels (SOA, SCA, JBI, BPEL, J2EE, Web, CORBA, Fractal, bases de données, etc.). Pour cela, FDF est composé d'un langage de déploiement, d'une bibliothèque de composants de déploiement et d'un outil graphique pour gérer les déploiements. Le langage FDF est un langage de description d'architectures permettant de décrire des déploiements, c'est-à-dire l'ensemble des machines cibles et les logiciels à déployer sur celles-ci. Ce langage permet de composer des composants de déploiement qui encapsulent les différents protocoles de transferts de fichiers (*e.g.*, FTP, HTTP, SCP), d'accès à distance (*e.g.*, TELNET, SSH), les shells Unix et Windows shells, les notions relatives à Internet (*e.g.*, nom de machine, port) et logiciels (*e.g.*, intergiciels, services, démons, serveurs d'applications, composants applicatifs). FDF a été intégré dans trois produits industriels chez Bull et ScalAgent Distributed Technologies. Dans le papier [45] publié à la conférence CCGRID 2008, nous avons validé FDF sur le déploiement d'un système distribué sur mille noeuds de Grid 50000, l'infrastructure de grille expérimentale française dédiée à la recherche en informatique. A l'époque, c'était le plus large déploiement automatiquement réalisé.

Ce travail contribue à mes deux questions de recherche. FDF offre un langage pour exprimer le déploiement (QR1) et un intergiciel à base de composants pour effectuer le déploiement (QR2).

Composants Java temps-réels De 2006 à 2009, j'ai co-encadré avec le Prof. Lionel Seinturier la thèse de Ales Plšek portant sur une approche intégrée nommée SOLEIL pour la conception et le développement de systèmes temps-réels Java à base de composants FRACTAL [190]. SOLEIL propose une approche dirigée par les modèles (MDE) pour la conception d'architectures logicielles temps-réels. Cette approche permet d'isoler le code métier et le code spécifique aux propriétés temps-réels. Le code métier est encapsulé dans des composants FRACTAL classiques. Le code temps-réel est pris en charge par des membranes de composants, c'est-à-dire des conteneurs temps-réels pour les composants métiers. Ce travail a donné lieu à plusieurs publications [193, 192, 89] dont la principale est notre papier à la conférence Middleware 2008 [191] qui présente notre modèle de composants pour des systèmes embarqués temps-réels en Java.

Ce travail contribue principalement à ma deuxième question de recherche (QR2). SOLEIL propose un modèle de composants pour les systèmes embarqués temps-réels Java.

Formalisation du modèle de composants Fractal En 2008, j'ai contribué avec Jean-Bernard Stefani à la formalisation du modèle de composants FRACTAL avec le langage de spécification formelle ALLOY [136]. Sur cette base, nous avons proposé en 2010 le langage formel FRACTOY pour la spécification de systèmes auto-configurables à base de composants [223].

5. `http://fdf.gforge.inria.fr`

Ces travaux contribuent à ma première question de recherche (QR1).

1.3.4 Le modèle Services Composants Aspects de l'intergiciel d'intergiciels FraSCAti (2007 - 2015)

Cette quatrième et dernière période regroupe mes travaux sur le modèle Services Composants Aspects (SCA) de l'intergiciel d'intergiciels FraSCAti.

FraSCAti Depuis 2007, je suis l'initiateur et le responsable du projet de recherche FraSCAti en collaboration avec le Prof. Lionel Seinturier et Romain Rouvoy. Le projet FraSCAti s'attaque à l'adaptabilité dans les architectures orientées services (SOA) à travers deux défis scientifiques : 1) l'adaptabilité face à l'hétérogénéité / variabilité des paradigmes d'interaction (invocation à distance, orienté message, orienté événement, groupe, etc.), des protocoles de communication (SOAP, JMS, OMG DDS, JGroups, etc.), des langages de description de services (WSDL, WADL, IDL, etc.), des langages et canevas d'implantation (Java, WS-BPEL, OSGi, etc.) et des environnements d'exécution (senseur, téléphone, tablette, station de travail, infrastructure de cloud, etc.) et 2) l'adaptabilité à l'exécution permettant l'auto-configuration et l'auto-optimisation en fonction du contexte d'exécution, la maintenance à chaud minimisant l'indisponibilité des systèmes logiciels et la reconfiguration logicielle aussi bien structurelle que comportementale.

La contribution scientifique du projet FraSCAti est de proposer un intergiciel réflexif pour l'informatique orientée service combinant deux idées originales : sa notion d'intergiciel d'intergiciels et son modèle Services Composants Aspects réflexif. Partant du constat qu'il n'existe pas d'intergiciel universel capable de couvrir l'ensemble des besoins de toutes les applications distribuées, le projet FraSCAti propose un canevas intergiciel extensible pour l'intégration et la composition élégante des intergiciels et technologies SOA existants. Le modèle SCA réflexif du projet FraSCAti est quant à lui le mariage fécond du standard OASIS *Service Component Architecture* (SCA), du modèle de composants Fractal et de la programmation orientée aspects (AOP). Dans ce modèle, tout est composant réflexif permettant ainsi d'adapter dynamiquement aussi bien les applications métiers, l'intergiciel, les liaisons de communication réseau que les aspects non fonctionnels. Cette réflexivité comprend l'introspection et la reconfiguration structurelle et comportementale. Ma recherche s'inscrit ainsi dans la continuité des travaux de Jean-Bernard Stefani, père du modèle de composants Fractal qui a fortement influencé les communautés intergiciel et composant, que j'ai étendus au domaine des architectures orientées services.

Le projet FraSCAti a donné lieu à deux publications majeures. Le papier [213] a été publié en 2009 dans la principale conférence internationale IEEE en Service Computing. Ce papier est le 2ème papier SCC le plus cité sur la période 2009 - 2014 en totalisant 122 citations au 1 juin 2015. Une version étendue [214] a été publiée en 2012 dans le journal Software : Practice and Experience (SPE). Cet article est le 5ième article SPE le plus cité sur la période 2009 - 2014 en totalisant 110 citations au 1 juin 2015.

Parallèlement, le projet FraSCAti a donné lieu au développement du logiciel FraSCAti hébergé par le consortium international OW2 [6]. Ce logiciel implante un intergiciel réflexif pour l'informatique orientée service, la notion d'intergiciel d'intergiciels et le modèle SCA réflexif. Ce logiciel est une des principales implantations en logiciel libre (licence LGPL) du standard

6. `http://frascati.ow2.org`

OASIS Service Component Architecture (SCA). Le logiciel FraSCAti est plus efficace que son principal concurrent Apache Tuscany développé par IBM et intégré à leur produit WebSphere Application Server (voir [214]). Le logiciel FraSCAti totalise plus de 292 000 lignes effectives de code. Je suis le principal concepteur, architecte et contributeur (développement, maintenance et extension) du logiciel FraSCAti avec environ 30% des commits. A ma connaissance, le logiciel FraSCAti est intégré dans huit logiciels d'industriels : 6 chez la société Linagora (OW2 Petals, EasyViper, EasyBPEL, EasyESB, EasierCos, EasierBSM) et 2 chez la société Open Wide SA (EasySOA Registry, OW2 Scarbo). En plus des contrats auxquelles j'ai participé, le logiciel FraSCAti est transitivement embarqué et utilisé dans la couche intergiciel des projets FP7 ICT FET IP CHOReOS, ANR SocEDA, FUI PICOM Macchiato, FUI PICOM CAPPUCINO et ARC SERUS. Quatorze thèses ont choisi le logiciel FraSCAti comme plate-forme d'exécution et celui-ci est utilisé dans des travaux de recherche aussi bien en France (Inria ADAM et OBASCO, LAAS-TSF), en Espagne, en Suède (KTH) et au Canada (UQàM). Ce logiciel est aussi utilisé en enseignement en France, au Brésil et en Colombie (U. de Cali et U. de Los Andes).

Le projet FraSCAti présenté plus en détails dans le Chapitre 2 contribue à mes deux questions de recherche en proposant le paradigme SCA pour la construction des applications réparties (QR1) et l'organisation d'un intergiciel d'intergiciels (QR2).

Orchestration de services à large échelle De 2008 à 2011, j'ai co-encadré avec la Prof. Françoise Baude de l'équipe-projet Inria OASIS à l'Université de Nice - Sophia Antipolis la thèse de Virginie Legrand Contes [87]. Cette thèse a été menée dans le contexte du projet européen SOA4All [7]. Cette thèse propose une approche à composants SCA pour l'orchestration de services à large échelle [87]. L'idée principale de cette thèse est de décomposer un processus métier en un ensemble de sous-processus selon des contraintes spatio-temporelles et de distribuer l'exécution de ces sous-processus. Cette thèse est présentée plus en détails dans la section 2.10.

Ce travail contribue à ma deuxième question de recherche (QR2) en proposant une approche à composants pour l'orchestration de services à large échelle.

Évolution des Systèmes de Systèmes De 2009 à 2012, j'ai co-encadré avec le Prof. Lionel Seinturier la thèse CIFRE de Jonathan Labéjof financée par la société Thales Communications & Security. Cette collaboration s'est inscrite dans le projet ANR ITEmIS [8] portant sur l'intégration des systèmes orientés services avec les systèmes embarqués. Cette thèse a étudié la réflexivité au service de l'évolution des systèmes de systèmes [80] et adresse plus particulièrement l'interopérabilité et l'adaptabilité des intergiciels orientés messages et orientés données de plus en plus utilisés dans les systèmes embarqués. Cette thèse propose trois contributions. R-DDS adresse la reconfigurabilité des intergiciels conformes au standard OMG Data Distribution Service (DDS) aussi bien au niveau métier que extra-fonctionnel. R-DDS est l'objet d'un brevet [79] déposé en Europe et aux U.S.A, qui sera exploité dans de futurs produits chez Thales. R-MOM adresse l'interopérabilité entre intergiciels asynchrones (MOM) aussi bien au niveau des protocoles réseaux que des personnalités applicatives (API) [78]. R-EMS adresse la spécification de systèmes de systèmes évolutifs. Cette thèse est présentée plus en détails dans la section 2.11.

7. `http://cordis.europa.eu/fp7/ict/ssai/docs/fp7call1achievements/soa4all.pdf`
8. `https://research.linagora.com/display/itemis/ITEmIS+Overview/`

Ce travail contribue à ma deuxième question de recherche (QR2) en proposant une approche à composants réflexifs pour la construction d'intergiciels asynchrones.

Plate-forme multi-nuages De 2011 à 2014, j'ai co-encadré la thèse de Fawaz Paraiso [174] avec le Prof. Lionel Seinturier. Cette thèse s'inscrit dans le domaine de recherche en plein essor du *Multi-Cloud Computing*. Face à l'hétérogénéité des nuages, nous proposons SOCLOUD une plate-forme multi-nuages distribuée pour la conception, le déploiement et l'exécution d'applications distribuées à large échelle. SOCLOUD adresse la portabilité, le déploiement, l'élasticité et la haute disponibilité d'applications multi-nuages [178]. Cette thèse est présentée plus en détails dans la section 2.12.

Ce travail contribue à mes deux questions de recherche en proposant le paradigme SCA pour la construction d'applications multi-nuages (QR1) et l'organisation en composants SCA de l'intergiciel multi-nuages SOCLOUD (QR2).

1.4 Organisation du manuscrit

Le Chapitre 1 a présenté une synthèse de mon parcours professionnel, de mon domaine de recherche et de mes principales contributions depuis 1997.

Le Chapitre 2 est consacré aux travaux menés depuis 2007 sur l'intergiciel d'intergiciels FRASCATI, le modèle Services Composants Aspects (SCA) et les trois dernières thèses que j'ai co-encadrées.

Le Chapitre 3 dresse un bilan de mon activité de recherche et discute de mes perspectives de recherche.

Chapitre 2

Adaptabilité de services distribués

Sommaire

2.1 Introduction historique . . . 14
2.2 Un rapide survol du modèle SCA . . . 15
2.2.1 La genèse de SCA . . . 15
2.2.2 Le modèle d'assemblage SCA . . . 16
2.2.3 L'application hétérogène Twitter - Weather . . . 18
2.3 Questions de recherche . . . 19
2.4 La genèse de FraSCAti . . . 21
2.5 QR1 : Supporter la flexibilité / extensibilité / adaptabilité du modèle SCA . . . 23
2.5.1 Des composants partout . . . 23
2.5.2 Un intergiciel d'intergiciels . . . 24
2.5.3 L'intégration de multiples implantations . . . 26
2.5.3.1 Le cas de l'intégration du langage WS-BPEL . . . 26
2.5.4 L'interopérabilité de multiples liaisons . . . 28
2.5.4.1 Les composants de liaison . . . 29
2.5.5 Les besoins extra-fonctionnels . . . 30
2.5.5.1 Les composants d'aspect . . . 30
2.5.5.2 La capture de diagramme de séquence UML . . . 32
2.5.5.3 Les aspects de liaisons Apache CXF . . . 33
2.5.5.4 Model Driven Security@run.time . . . 35
2.5.6 La plate-forme FraSCAti . . . 38
2.5.7 Synthèse sur QR1 . . . 40
2.6 QR2 : Modulariser la flexibilité / extensibilité / adaptabilité d'une plate-forme SCA . . . 41
2.6.1 Une plate-forme modulable " à la carte " . . . 42
2.6.2 La fragmentation de la description de l'architecture de la plate-forme 43
2.6.3 La recomposition de l'architecture de la plate-forme FraSCAti . . 44
2.6.4 Synthèse et perspectives sur QR2 . . . 45
2.7 QR3 : Reconfigurer à l'exécution des applications SCA . . . 46
2.7.1 Equiper les composants de capacités réflexives . . . 46
2.7.2 L'interface réflexive de FraSCAti . . . 47
2.7.3 L'outil FraSCAti Explorer . . . 49

2.7.4 L'outil FraSCAti JMX 49
2.7.5 L'interface FraSCAti REST 50
2.7.6 La console FraSCAti Web Explorer 50
2.7.7 Le langage FraSCAti FScript 50
2.7.8 Bilan et perspectives sur QR3 53
2.8 QR4 : Modéliser la variabilité de la plate-forme FraSCAti 54
2.8.1 La modélisation des fonctionnalités de la plate-forme FraSCAti . . 54
2.8.2 L'extraction automatique du modèle de fonctionnalités de la plate-forme FraSCAti 57
2.8.3 L'évolution du modèle de fonctionnalités de la plate-forme FraSCAti 61
2.9 Synthèse sur FraSCAti 62
2.10 L'orchestration décentralisée et dynamique de processus métiers à base de services 66
2.10.1 Le contexte 66
2.10.2 La problématique 67
2.10.3 Les contributions 68
2.11 La réflexivité au service de l'évolution des systèmes de systèmes . 71
2.11.1 Le contexte et la problématique 71
2.11.2 Les contributions 72
2.11.2.1 R-DDS : la reconfigurabilité de l'intergiciel DDS 72
2.11.2.2 R-MOM : l'interopérabilité des intergiciels asynchrones . . 75
2.11.2.3 R-EMS : la spécification de systèmes de systèmes 76
2.12 La plate-forme multi-nuages soCloud 77
2.12.1 Le contexte et la problématique 77
2.12.2 Les contributions 79
2.12.2.1 Le modèle soCloud 79
2.12.2.2 La plate-forme soCloud 81
2.12.2.3 Trois applications soCloud 82

2.1 Introduction historique

Ce chapitre présente mes travaux de recherche sur l'**adaptabilité de services distribués** menés sur la période 2007 à 2015 au sein de l'équipe de recherche Inria ADAM devenue Spirals. Durant cette période, je me suis intéressé à l'adaptabilité en informatique orientée service (SOC [1]) qui " *promeut l'idée d'assembler des composants applicatifs en un réseau de services* [2] *[171]* " [171]. Dans l'article précité et sa version étendue [172], Michael P. Papazoglou et al dressent un état de l'art et une feuille de route de la recherche en SOC. Nos travaux et plus particulièrement ceux présentés à la section 2.7 ont contribué à résoudre le premier défi de recherche que ces auteurs ont identifié, à savoir **architectures reconfigurables dynamiquement à l'exécution** [3] **[172]**. Plus récemment, Valérie Issarny et al ont identifié l'**hétérogénéité** comme l'un des principaux défis de recherche à traiter dans les intergiciels orientés services pour l'Internet du futur [68]. Nos travaux contribuent à résoudre ce défi, plus

1. *Service-Oriented Computing*
2. *"Service-oriented computing promotes the idea of assembling application components into a network of services"*
3. *"dynamically reconfigurable runtime architectures"*

particulièrement dans le cadre du modèle *Service Component Architecture* (SCA) présenté dans la section 2.2 et de notre intergiciel d'intergiciels FraSCAti.

De 2007 à 2009, j'ai été le responsable scientifique du projet ANR SCOrWare dont l'objectif fut de construire une plate-forme ouverte orientée services à base de composants. Ce projet a produit deux résultats majeurs : 1) le support d'exécution SCA nommé FraSCAti [4] [51] et 2) l'environnement de modélisation pour SCA nommé Eclipse SCA Tools [5] [32]. Les principaux traits de conception et de réalisation de FraSCAti sont décrits dans la suite de ce chapitre.

De 2008 à 2011, j'ai participé au projet européen SOA4All [6] portant sur les architectures orientées services pour tous. Dans le cadre de ce projet, j'ai co-encadré la thèse de Virginie Legrand Contes avec la Prof. Françoise Baude de l'Université de Nice - Sophia Antipolis. Cette thèse propose une approche à composant pour l'orchestration de services à large échelle [87]. Cette approche est résumée dans la section 2.10.

De 2009 à 2012, j'ai participé au projet ANR ITEmIS [7] portant sur les systèmes intégrés orientés services et embarqués. Dans le cadre de ce projet, j'ai co-encadré avec le Prof. Lionel Seinturier la thèse CIFRE de Jonathan Labéjof financée par la société Thales Communications & Security. Cette thèse étudie la réflexivité au service de l'évolution des systèmes de systèmes [80]. Les résultats principaux de cette thèse sont décrits dans la section 2.11.

De 2011 à 2014, j'ai co-encadré la thèse de Fawaz Paraiso avec le Prof. Lionel Seinturier. Cette thèse propose soCloud : une plate-forme multi-nuages distribuée pour la conception, le déploiement et l'exécution d'applications distribuées à large échelle [174]. Cette plate-forme est présentée dans la section 2.12.

2.2 Un rapide survol du modèle SCA

2.2.1 La genèse de SCA

Service Component Architecture (SCA) [90] est un ensemble de spécifications pour développer et déployer des applications composites s'inscrivant nativement dans une architecture orientée service (SOA) indépendamment des produits, des plates-formes, des intergiciels, des protocoles de communication réseau, des politiques extra fonctionnelles et des langages utilisés.

Une application fondée sur SCA est un assemblage de composants logiciels. Chaque composant implémente une partie de la logique métier, peut dépendre de services externes et peut à son tour exposer tout ou une partie de son comportement sous la forme de services. Le choix de la technologie à services à mettre en oeuvre est effectué lors de l'assemblage des composants et non lors de la programmation de chaque composant. Ainsi le code métier est découplé des détails techniques de mise en oeuvre des services rendant ainsi la logique métier réutilisable dans différentes technologies à services comme les Web Services [6] ou bien le style architectural REST [44]. De plus, la logique métier peut être écrite dans différents langages de programmation tels que Java, C++, WS-BPEL, COBOL, etc.

Les spécifications SCA ont été initialement rédigées courant 2006 par l'OSOA Collaboration, un regroupement d'industriels du logiciel incluant BEA Systems, IBM, IONA

4. http://frascati.ow2.org/
5. http://www.eclipse.org/soa/sca/
6. http://cordis.europa.eu/fp7/ict/ssai/docs/fp7call1achievements/soa4all.pdf
7. https://research.linagora.com/display/itemis/ITEmIS+Overview/

Technologies, Oracle Corporation, SAP AG, Sybase, Xcalia, Zend Technologies, Cape Clear, Interface21, Primeton Technologies, Progress Software, Red Hat, Rogue Wave Software, Software AG, Sun Microsystems, TIBCO Software et Siemens AG. La version 1.0 de SCA a été publiée en mars 2007 [12]. A partir de 2007, les spécifications SCA ont été endossées par le consortium international de standardisation OASIS [8]. En parallèle, nous avons soumis le projet ANR SCOrWare courant 2006, ce projet a débuté en janvier 2007 et le premier prototype de FraSCAti est sorti durant l'été 2007. Ainsi nous étions tout à fait en phase avec les préoccupations d'innovations technologiques et industrielles du moment.

Après de nombreuses années d'incubation, quelques implantations de SCA en logiciel libre sont maintenant disponibles telles que Apache Tuscany [9], OW2 FraSCAti [10], JBoss SwitchYard [11], Fabric3 [12], Service Conduit [13] et Trentino [14]. Dans le monde industriel, nous pouvons citer le produit IBM WebSphere Application Server incluant Apache Tuscany comme noyau d'exécution SCA ainsi que les outils de modélisation et de développement du projet Eclipse SCA Tools.

2.2.2 Le modèle d'assemblage SCA

Le modèle d'assemblage de SCA comprend 9 concepts clés comme illustrés par la figure 2.1 :

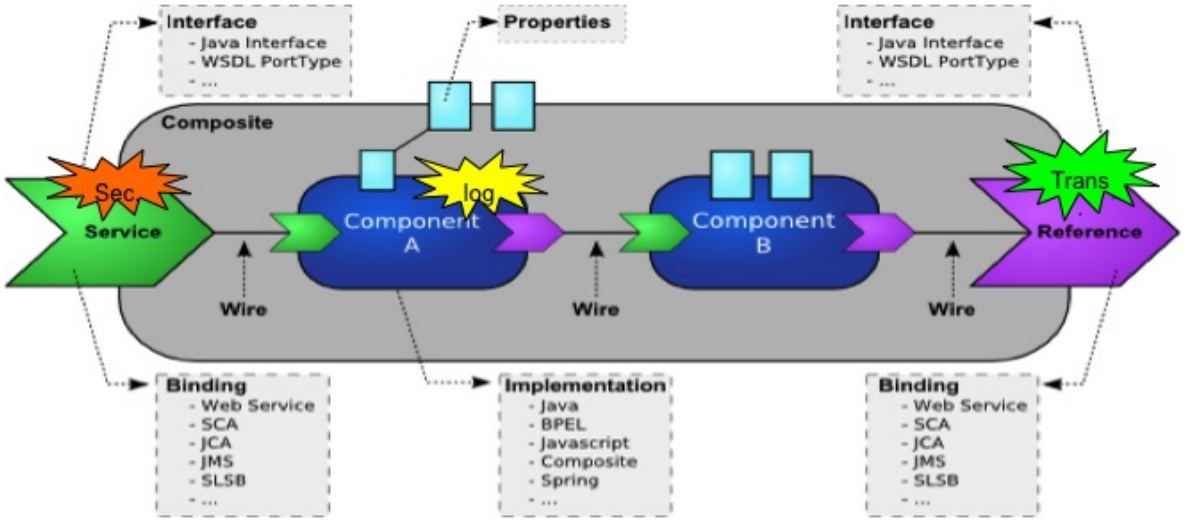

Figure 2.1 – Un composite SCA.

- **Composite** est le point d'entrée d'un assemblage ou application. Un composite contient un ou des composants SCA implantant la logique métier de l'application. Un composite

8. http://www.oasis-opencsa.org/sca/
9. http://tuscany.apache.org
10. http://frascati.ow2.org/
11. http://switchyard.jboss.org/
12. http://www.fabric3.org/
13. http://www.service-conduit.org/
14. http://sourceforge.net/projects/trentino/

peut exporter des propriétés de ces sous-composants, exposer des services de ces sous-composants et requérir des références pour ces sous-composants.

- **Component** encapsule toute ou un fragment de la logique métier d'une application. Un composant est une instance d'une implantation. Celle-ci est configurée par des propriétés. Un composant offre des fonctionnalités métiers via des services et peut requérir des services externes via des références configurables.
- **Implementation** est une pièce de code implantant la logique métier d'un composant. Ici, SCA est flexible car il ne contraint pas la nature d'une implantation. Le consortium OASIS a standardisé l'implantation de composants SCA via des composites SCA [153] ou écrite en Java [155, 154], EJB [150], C [148], C++ [149] et BPEL [145]. Toutefois, une plate-forme SCA peut supporter d'autres formes d'implantation.
- **Property** permet de configurer la logique métier d'un composant ou d'un composite. Une propriété a un nom localement unique au sein d'un composant et sa valeur est typée.
- **Service** représente un point d'accès pour invoquer les fonctionnalités métiers d'un composant ou composite. Un service a un nom localement unique et est typé par une interface.
- **Reference** représente un besoin fonctionnel d'un composant ou composite. Une référence a un nom localement unique et est typée par une interface.
- **Interface** est une liste de signatures métiers offertes par un service ou requises par une référence. Ici, SCA est flexible car il ne contraint par la nature du langage décrivant une interface. Le consortium OASIS a standardisé la description d'interfaces SCA via WSDL [153] et les interfaces Java [153]. Toutefois, une plate-forme SCA peut supporter d'autres langages de description d'interfaces tels que OMG IDL [162], WADL [239], etc.
- **Wire** est une liaison entre une référence et un service au sein d'un même composite. Ces liaisons permettent donc d'assembler fonctionnellement les composants formant une application SCA.
- **Binding** est une liaison d'interopérabilité vers le monde extérieur. Une liaison placée sur un service permet d'exporter ce service dans un monde technologique donné, par exemple exposer un service en Web Services ou sous la forme d'une ressource Web. Une liaison placée sur une référence permet de lier cette référence à un service externe, par exemple se connecter à un Web Service ou une ressource REST pré-existants. Le consortium OASIS a standardisé des liaisons pour les Web Services [159], Java Messaging Service (JMS) [156], Java EE Connector Architecture (JCA) [151]. Toutefois, une plate-forme SCA peut supporter d'autres liaisons d'interopérabilité tels que Java RMI, CORBA, REST, etc.

Orthogonalement, SCA permet d'associer des propriétés extra-fonctionnelles telles que la sécurité, les transactions, le monitoring, à tout composite, composant, implantation, service, référence et liaison. Ici, SCA est flexible car il ne contraint par la nature des propriétés extra-fonctionnelles prises en charge. Pour cela, SCA fournit un *Policy Framework* [152] pour spécifier ces propriétés extra-fonctionnelles désignées sous le terme d'*intent* dans le langage SCA. Ce canevas spécifie des intentions pour la sécurité (authentification, confidentialité, intégrité), la fiabilité et la démarcation des transactions. Toutefois, une implantation de SCA est libre de supporter d'autres politiques extra-fonctionnelles.

Finalement, SCA offre la notion de contribution pour conditionner des applications SCA au sein d'une archive déployable. Par défaut, une contribution SCA est une archive au format

ZIP.

2.2.3 L'application hétérogène Twitter - Weather

SCA permet de mixer des technologies hétérogènes au sein d'une même application comme l'illustre l'application TWITTER - WEATHER de la figure 2.2, qui fut publiée dans Programmez ! le magazine du développeur [15] [30, 29]. Cette application " jouet " offre un service pour obtenir la météo de la ville où vit un utilisateur TWITTER. Ce service est offert en tant que service Web ainsi que comme une ressource REST. La logique métier est réalisée par un processus WS-BPEL et une classe JAVA utilitaire. Le service TWITTER est accessible via une interface REST tandis que le service de météo est accessible via une interface WSDL [16].

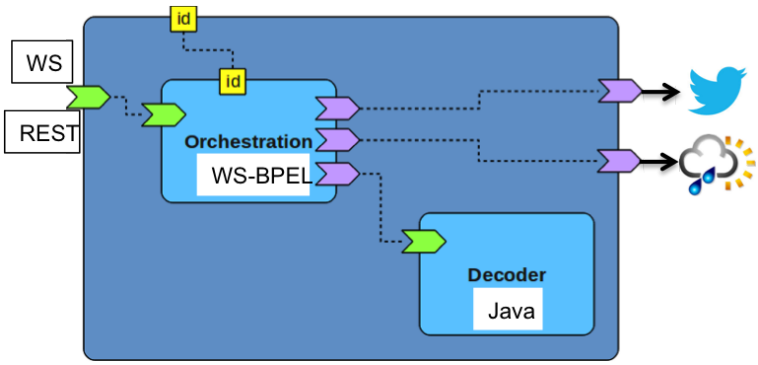

FIGURE 2.2 – L'application TWITTER - WEATHER.

Pratiquement, le modèle d'assemblage SCA est supporté par un langage dédié exprimé en XML. Le listing 2.1 présente l'assemblage SCA pour l'application TWITTER - WEATHER.

Le service de cette application est spécifié par les lignes 2 à 5. Il est exporté comme un service Web (ligne 3) ainsi qu'une resource REST (ligne 4). Le coeur de la logique métier est contenu dans le composant `Orchestration` (lignes 6 à 9). En ligne 8, il référence le service du composant `Decoder` (lignes 10 à 15). La référence sur le service TWITTER est matérialisée par les lignes 16 à 18. La référence vers le service de météo est exprimée par les lignes 19 à 22. Les attributes `promote` en lignes 2, 16 et 19 permettent de lier les services / références du composite avec celles de ces sous-composants. L'attribut `require` en ligne 6 permet d'exprimer l'intention de tracer toutes les interactions du composant `Orchestration`. Sans entrer dans les détails, certaines informations peuvent ne pas être explicitement spécifiées et alors elles sont inférées depuis l'implantation des composants comme par exemple l'interface du service `tw` et des références `twitter` et `weather`, le service offert du composant `Orchestration`, etc. Une implantation complète de cette application est disponible dans la base de code de FRASCATI [17].

```
1 <composite name="twitter-weather">
2   <service name="tw" promote="Orchestration">
3     <binding.ws uri="/ws/TwitterWeather"/>
4     <binding.rest uri="/rest/TwitterWeather"/>
5   </service>
6   <component name="Orchestration" require="logging">
```

15. http://www.programmez.com/
16. *Web Service Description Language*
17. http://websvn.ow2.org/listing.php?repname=frascati&path=/trunk/examples/twitter-weather/

```
7       <implementation.bpel process="orchestration.bpel"/>
8       <reference name="decoder" target="Decoder/decode"/>
9     </component>
10    <component name="Decoder">
11     <implementation.java class="DecoderImpl"/>
12     <service name="decode">
13       <interface.java interface="Decoder"/>
14     </service>
15    </component>
16    <reference name="twitter" promote="Orchestration/twitter">
17      <binding.rest uri="https://twitter.com"/>
18    </reference>
19    <reference name="weather" promote="Orchestration/weather">
20      <binding.ws wsdlLocation="http://www.webservicex.net/globalweather.asmx?wsdl"
21                  wsdlElement="http://www.webservicex.net#wsdl.port(GlobalWeather/
                        GlobalWeatherSoap)"/>
22    </reference>
23  </composite>
```

Listing 2.1 – La description SCA simplifiée de l'application TWITTER - WEATHER

Cette application illustre parfaitement que **SCA offre la capacité d'exprimer des orchestrations de services hétérogènes distribués**. Dans la section 2.10, la thèse de Virginie Legrand Contes montre comment ces orchestrations peuvent elles-mêmes être distribuées.

2.3 Questions de recherche

Comme nous venons de le voir précédemment, SCA est un standard OASIS pour bâtir simplement des applications orientées services à base de composants logiciels. Technologiquement agnostique, le standard SCA offre nativement **quatre points de flexibilité** au niveau du choix de la nature de l'implantation de chaque composant, du langage de description des interfaces, des technologies de liaison et des propriétés extra-fonctionnelles. Ces points de flexibilité peuvent être vu comme des **points d'extensibilité** du standard SCA mais aussi comme des **points d'adaptabilité** à la conception des applications. Choisir la nature de l'implantation de chaque composant permet de supporter l'**intégration** de codes patrimoniaux. Choisir la nature de la technologie de liaison permet de supporter l'**interopérabilité** avec des applications non nativement SCA telles que la majeure partie des services accessibles sur Internet (e.g., AMAZON, GOOGLE, etc.). Ainsi **une plate-forme implantant le standard SCA doit nativement supporter au moins les quatre points de flexibilité / extensibilité / adaptabilité du standard SCA**. Dès lors, plusieurs questions de recherche se posent comme illustré par la figure 2.3 :

- **QR1 : comment supporter la flexibilité / extensibilité / adaptabilité du modèle SCA ?** Cela revient à proposer une plate-forme intergicielle offrant la flexibilité prédéfinie du modèle SCA et une extensibilité non anticipée à l'avance pour permettre de construire des applications SCA adaptables à la conception en termes d'intégration, d'interopérabilité et de besoins extra-fonctionnels.
- **QR2 : comment modulariser la flexibilité / extensibilité / adaptabilité d'une plate-forme SCA ?** Bien qu'une plate-forme SCA doive supporter différentes technologies d'implantation des composants / des liaisons, différents langages de programmation et de description d'interfaces, diverses propriétés extra-fonctionnelles, une même application métier nécessite rarement l'ensemble de ces technologies simultanément. Par

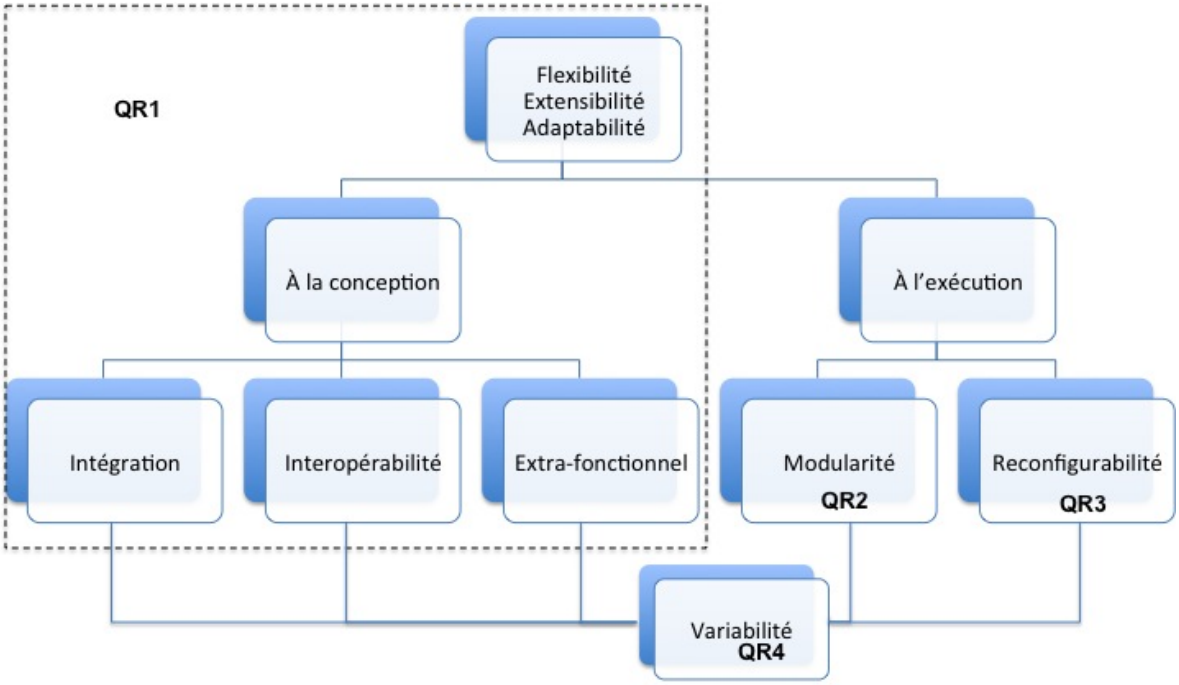

FIGURE 2.3 – Quatre questions de recherche pour l'adaptabilité de services distribués.

exemple, toutes les applications n'ont pas besoin d'un moteur d'exécution WS-BPEL, seules celles qui nécessitent des mécanismes de compensation contiendront des implantations en WS-BPEL. De même, une application embarquée limitera les technologies utilisées afin de réduire son empreinte mémoire, disque et/ou CPU. Ainsi, il convient de proposer une plate-forme intergicielle modulable " à la carte ", c'est-à-dire que les fonctionnalités réellement présentes à l'exécution doivent pouvoir être sélectionnées en fonction des besoins réels des applications à exécuter.

- **QR3 : comment reconfigurer à l'exécution des applications SCA ?** Le standard SCA définit un langage dédié XML pour assembler et configurer des composants pour former des applications orientées services. Ce langage est utilisé durant la conception des applications et est interprété au moment du déploiement de celles-ci. Toutefois, le standard SCA ne spécifie pas les moyens pour reconfigurer les applications durant leur exécution une fois que celles-ci ont été déployées. L'ajout de tels mécanismes de reconfiguration au standard SCA permettrait d'envisager des applications orientées services dynamiquement adaptables voire auto-adaptables.
- **QR4 : comment modéliser la variabilité d'une plate-forme SCA ?** Une plate-forme SCA doit être adaptable et extensible pour supporter des applications adaptables et reconfigurables. Ainsi une telle plate-forme va offrir un très grand nombre de fonctionnalités. Ces fonctionnalités peuvent être compatibles entre elles (ou non) et des relations de dépendances peuvent exister entre ces fonctionnalités. Il convient donc de pouvoir modéliser cette variabilité.

Les sections suivantes présentent les réponses apportées à ces quatre questions de recherche lors de nos travaux sur la plate-forme FRASCATI.

2.4 La genèse de FraSCAti

Au cours d'une recherche bibliographique courant 2006, nous sommes tombés sur des documents de travail de l'OSOA Collaboration spécifiant les prémisses de SCA. Rapidement, nous avons identifié le potentiel de SCA et avons décidé de monter un projet ANR. Début janvier 2007, le projet ANR SCOrWare a démarré avec comme objectif de construire une plate-forme ouverte orientée services à base de composants. Ce projet avait trois sous-buts : 1) étudier les relations entre architecture orientée service (SOA) et les approches architecturales à base de composants, plus spécifiquement étudier les similarités et différences entre les modèles *Service Component Architecture* (SCA), *Java Business Integration* (JBI) et le modèle Fractal, 2) développer une implantation à base de composants de la spécification SCA et 3) développer des outils de modélisation et de développement de haut niveau pour aider les utilisateurs finaux à adopter cette nouvelle technologie. Les partenaires du projet étaient Artenum, EBM WebSourcing, Inria (équipes ADAM, ObjectWeb et Sardes), INT, IRIT, Obeo et Open Wide. Je fus le responsable scientifique de ce projet d'une durée de deux ans. Ce projet a produit trois résultats majeurs : 1) un support d'exécution pour SCA [51], 2) un environnement dirigé par le modèle SCA [32] et 3) des démonstrateurs industriels [16].

A l'été 2007, un premier prototype d'un support d'exécution pour SCA a été mis à disposition des partenaires du projet ANR SCOrWare. En janvier 2008, le projet FraSCAti [18] a été officiellement créé auprès du consortium international OW2. Depuis, je suis le responsable de ce projet OW2, son architecte principal et l'un des 30 co-développeurs (j'ai effectué environ 30% des commits [19]). La version 1.0 du logiciel FraSCAti est sortie en juin 2009. Maintenant, nous en sommes à la version 1.6 qui contient plus de 292 000 lignes de code [20], principalement du code Java et des assemblages SCA. Globalement, le nombre de téléchargements du logiciel FraSCAti est de plus de 47 064 depuis 2009. Il y a 64 inscrits sur la liste de diffusion FraSCAti. Le site du projet FraSCAti a été visité plus de 10 000 fois depuis août 2011.

Le logiciel libre FraSCAti a été intégré dans au moins huit logiciels libres d'industriels, principalement chez les sociétés Linagora (ex EBM WebSourcing) et Open Wide SA.

A l'issu du projet ANR SCOrWare, le bus d'entreprises distribué libre OW2 PEtALS [21] de la société EBM WebSourcing proposait un moteur de services SCA construit avec FraSCAti. Dans le cadre des projets ANR ARPEGE SALTY [22] (*Self-Adaptive very Large disTributed sYstems*) et SocEDA [23] (plate-forme EDA sociale, largement distribuée pour l'informatique dans les nuages), nous avons collaboré avec la société Linagora afin de porter ces prototypes logiciels de recherche au dessus de FraSCAti. Ainsi, son bus d'entreprises léger EasyESB [24], sa plate-forme de gouvernance EasyBSM [25] et ses moteurs d'exécution

18. http://frascati.ow2.org/
19. https://www.openhub.net/p/frascati/contributors/summary
20. https://www.openhub.net/p/frascati/analyses/latest/languages_summary
21. http://petals.ow2.org/
22. https://salty.unice.fr/
23. https://research.linagora.com/display/soceda/SocEDA+Overview/
24. https://research.petalslink.org/display/easyesb/EASYESB+-+Open+source+Lightweight+Services+Bus/
25. https://research.petalslink.org/display/easierbsm/EasierBSM+-+Open+source+Business+Service+Monitoring+and+SLA+enforcement/

de processus métiers – le moteur générique EasyViper [26], EasyBPEL [27] pour WS-BPEL, EasyCos [28] pour BPMN 2.0 – sont construits sous la forme d'assemblages de composants SCA s'exécutant au dessus de notre plate-forme FraSCAti. Comme la société Linagora réutilise ces prototypes dans d'autres projets de recherche, FraSCAti est ainsi transitivement embarqué dans les logiciels issus de ces projets. Par exemple, l'intergiciel du projet européen CHOReOS [29] s'appuie sur le bus léger EasyESB et donc transitivement sur FraSCAti.

A l'issu du projet ANR SCOrWare, le logiciel libre OW2 Scarbo [30] de la société Open Wide SA utilise la plate-forme de services FraSCAti. Dans le cadre du projet FUI EasySOA [31], j'ai collaboré avec cette société pour intégrer FraSCAti dans le logiciel libre EasySOA Registry [32].

FraSCAti a aussi été utilisé comme support d'exécution dans les projets Cappucino et Macchiato du pôle de compétitivité de l'industrie du commerce (PICOM). Le projet Cappucino visait la construction et adaptation d'applications ubiquitaires et de composants d'intergiciels en environnement ouvert pour l'industrie du commerce. Le projet Macchiato portait sur l'ouverture des systèmes d'information de l'industrie du commerce à l'Internet des choses et proposa le panier unique du e-commerce.

FraSCAti est aussi utilisé par de nombreux travaux de recherche à travers le monde. Pour l'en citer qu'un, le projet SOFA [33] pour Service Oriented Framework For Analysis de l'équipe LATECE dirigée par la Prof. Naouel Moha de l'Université du Québec à Montréal propose un canevas pour la spécification et la détection de patterns et anti-patterns dans les architectures orientées services. Comme l'illustre la figure 2.4, ce canevas est un assemblage de composants SCA s'exécutant au dessus de FraSCAti. Au dessus de SOFA, quatre approches ont été proposées pour détecter les anti-patterns dans des architectures à base de SCA (approche SODA [140, 170]), de services Web (SODA-W [169]), de ressources REST (SODA-R [167]) et de processus WS-BPEL (SODA-B [168]). SODA utilise un *intent* FraSCAti pour détecter dynamiquement certains anti-patterns et FraSCAti fut un cas d'étude pour l'approche SODA [170].

Les principales contributions scientifiques sur FraSCAti ont été publiées dans les actes de la conférence internationale IEEE sur *Services Computing* (SCC 2009) [213] puis en version étendue dans le journal *Software : Practice and Experience* en 2012 [214]. Cette série citée plus de 230 fois présente la plate-forme intergicielle à base de composants FraSCAti pour des architectures orientées service reconfigurables. En 2010, une démonstration sur la reconfigurabilité d'architectures SCA distribuées fut présentée lors de la conférence internationale ASE [109]. J'ai présenté en décembre 2011 un tutoriel sur " FraSCAti : un intergiciel d'intergiciels adaptable et réflexif " [135] lors de la conférence internationale *Middleware* à Lisbonne au Portugal. Ces contributions sont détaillées dans les sections suivantes.

26. https://research.petalslink.org/display/easyviper/EasyViper+Overview/
27. https://research.petalslink.org/display/easybpel/EasyBPEL+Overview/
28. https://research.petalslink.org/display/easiercos/EasierCos+Overview/
29. http://www.choreos.eu/
30. http://scarbo.ow2.org/
31. http://www.easysoa.org/
32. http://www.easysoa.org/welcome/download/
33. http://sofa.uqam.ca/

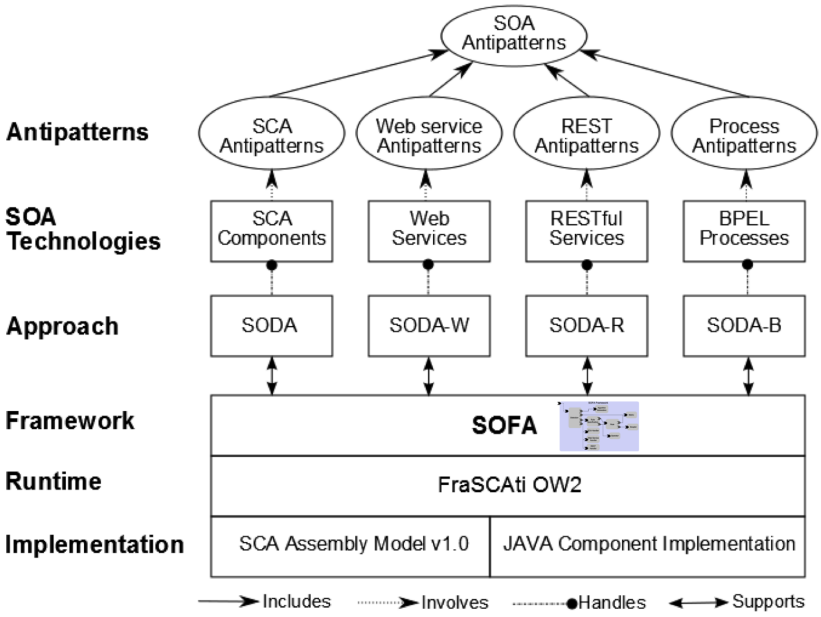

FIGURE 2.4 – L'organisation du canevas SOFA. Source : Francis Palma.

2.5 QR1 : Supporter la flexibilité / extensibilité / adaptabilité du modèle SCA

Cette section présente notre réponse à la première question de recherche (QR1) : comment supporter la flexibilité / extensibilité / adaptabilité du modèle SCA ? Cette question revient à proposer une plate-forme intergicielle offrant la flexibilité prédéfinie du modèle SCA et une extensibilité non anticipée à l'avance pour permettre de construire des applications SCA adaptables à la conception en termes d'intégration, d'interopérabilité et de besoins extra-fonctionnels.

Notre réponse pour supporter la flexibilité / extensibilité / adaptabilité du modèle SCA repose sur deux idées extrêmement simples mais puissantes : des composants partout et un intergiciel d'intergiciels.

2.5.1 Des composants partout

La première idée est que la notion de composant soit utilisée à tous les niveaux d'une architecture distribuée : bien sûr pour la logique métier (voir la section 2.2.2) mais aussi pour les technologies d'intégration / implantation des composants (voir la section 2.5.3), la réalisation des liaisons d'interopérabilité (voir la section 2.5.4), la réalisation des besoins extra-fonctionnels (voir la section 2.5.5) et l'architecture de la plate-forme d'exécution elle-même (voir la section 2.5.6). Cette idée s'inscrit dans une démarche visant à **promouvoir les composants comme entité unificatrice pour le développement de systèmes distribuées allant des couches applicatives aux couches intergicielles et systèmes**.

2.5.2 Un intergiciel d'intergiciels

Comme l'illustre la figure 2.5, l'intergiciel est une couche intermédiaire entre les applications (en haut) et les systèmes d'exploitation / réseau (en bas). Comme énoncé dans [76], l'intergiciel masque la distribution des applications et l'hétérogénéité des systèmes sous-jacents (OS, matériels et réseaux), tout en fournissant des abstractions et des interfaces de programmation de haut niveau pour structurer et développer les applications ainsi que des fonctions de base (interaction, annuaire, transaction, sécurité, etc.). Au niveau réseau, l'intergiciel fournit un protocole de transport des interactions entre les applications distribuées.

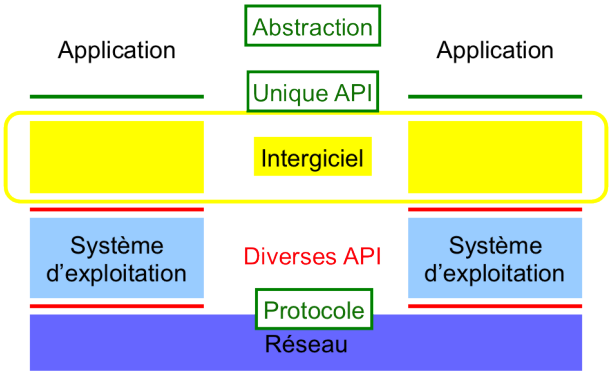

FIGURE 2.5 – La couche intergicielle intermédiaire entre les applications et les systèmes. Source : [13, 76].

Toutefois, il n'existe pas un intergiciel universel adéquate aux besoins de toutes les applications. Il existe plutôt une pléthore d'intergiciels se distinguant par la nature des interactions supportées (échange de messages, appel à distance, interaction synchrone ou asynchrone, interaction point à point ou multi-points, etc.), des abstractions fournies (mémoire partagée, procédure distante, objet distribué, composant, service, ressource, etc.), des langages de définition d'interfaces applicatives (OMG IDL, WSDL [34]) et des protocoles réseaux (CORBA/IIOP, SOAP [35], etc.). Au fil du temps, des intergiciels apparaissent, prospèrent puis disparaissent de la scène tels que par exemple le défunt *Distributed Computing Environment* (DCE), CORBA et les Web Services. Ainsi les architectes logiciels doivent s'interroger sur comment intégrer de nouveaux intergiciels prometteurs et supporter d'anciens intergiciels déclinants mais encore utilisés par des applications patrimoines. D'un autre côté, une application peut nécessiter de mettre en oeuvre différents paradigmes simultanément pour, par exemple, exposer certaines de ses fonctionnalités en tant que ressources Web, interagir avec des services Web tiers, interopérer avec des messageries et objets distants patrimoniaux. Ainsi **à l'origine l'intergiciel devait masquer l'hétérogénéité et maintenant nous sommes confrontés à une hétérogénéité d'intergiciels !**

Afin de résoudre ce problème d'hétérogénéité des intergiciels, divers travaux ont proposé des **intergiciels multi-personnalités**, c'est-à-dire des intergiciels implantant différentes interfaces de programmation et protocoles d'interaction simultanément. Historiquement, l'intergiciel QUATERWARE supportait trois personnalités : CORBA, JAVA RMI et MPI [216]. L'intergiciel JONATHAN supportait CORBA, JAVA RMI et des interactions multicast [41].

34. *Web Service Description Language*
35. *Simple Object Access Protocol*

Dans [228], l'intergiciel schizophrène PolyORB supportait plusieurs modèles de distribution tels que CORBA, SOAP, *Ada95 Distributed System Annex* (DSA), Ada Message Passing, Ada Web Server (AWS). Plus récemment, la pile logicielle Apache CXF très reconnue et utilisée dans l'industrie propose un intergiciel supportant les services Web et les ressources REST [36]. Toutefois, ces intergiciels implantent rarement un très grand nombre de personnalités hétérogènes car cela pose rapidement des problèmes de co-existence des personnalités et de maintenance de l'ensemble.

De notre côté, nous proposons la notion d'**intergiciel d'intergiciels** comme présenté lors de notre tutoriel à Middleware 2001 [135]. Un intergiciel d'intergiciels est un intergiciel intégrant d'autres intergiciels pré-existants et masquant leur hétérogénéité. Comme l'illustre la figure 2.6, FraSCAti est un intergiciel d'intergiciels. Vu des applications, FraSCAti présente une seule interface, à savoir le modèle d'assemblage de composants SCA, à la différence des intergiciels multi-personnalités. En interne, FraSCAti intègre différents supports d'exécution (voir section 2.5.3) et intergiciels d'interopérabilité (voir section 2.5.4).

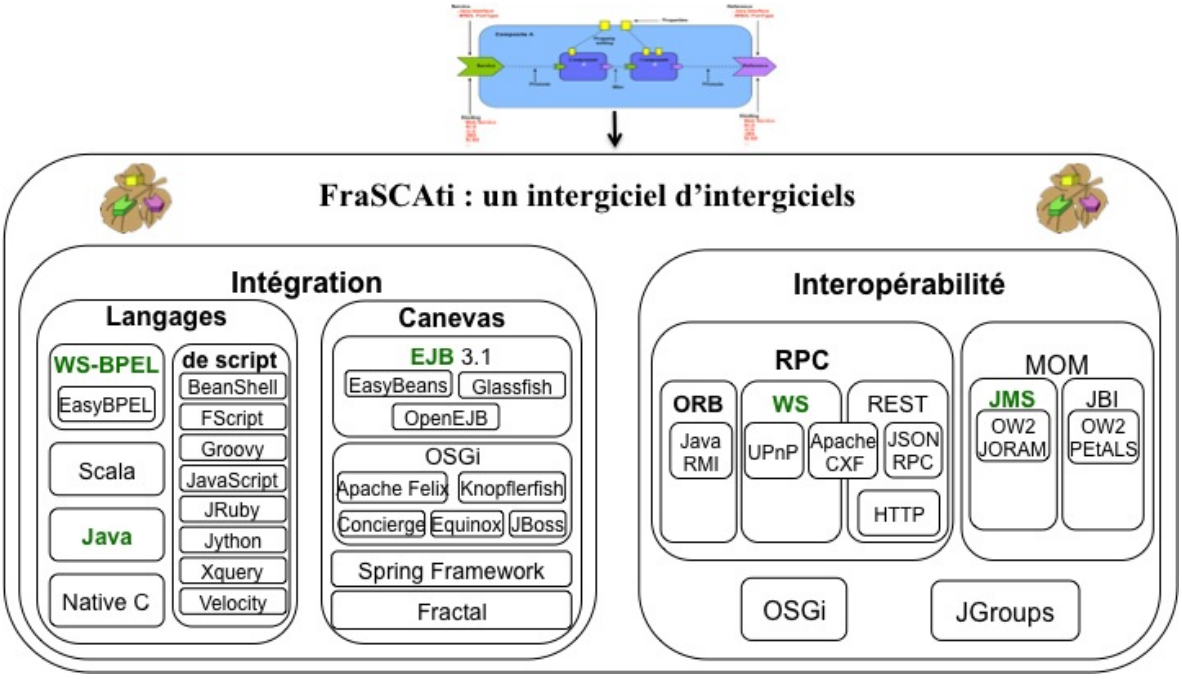

Figure 2.6 – FraSCAti : un intergiciel d'intergiciels.

La figure 2.6 montre clairement que FraSCAti va bien au déjà des spécifications SCA du consortium OASIS qui officiellement ne couvrent que l'implantation de composants en Java, WS-BPEL, EJB et des liaisons Web Services et JMS. De plus, on peut remarquer que FraSCAti intègre un nombre d'intergiciels plus significatif que les intergiciels multi-personnalités pré-cités.

36. `http://cxf.apache.org/`

2.5.3 L'intégration de multiples implantations

Conceptuellement en termes d'intégration, l'implantation de chaque composant peut être écrite avec n'importe quel langage de programmation ou réalisée via n'importe quel canevas de programmation.

Concrètement d'un coté, FraSCAti propose un support pour les langages de programmation C, Java, Scala, WS-BPEL ainsi que les langages de script BeanShell, FScript, Groovy, JavaScript, Ruby, Python, Xquery, et le langage de templates Velocity. Le cas du langage WS-BPEL est détaillé dans la section 2.5.3.1. D'un autre coté, FraSCAti intègre les canevas à composants Fractal, Spring Framework, OSGi et EJB 3.1. Différents moteurs d'exécution OSGi (Apache Felix, Concierge, Equinox, JBoss, Knopflerfish) et EJB (EasyBeans, Glassfish, OpenEJB) sont intégrés dans la plate-forme FraSCAti.

D'un coté, un développeur peut ainsi choisir librement pour chaque composant le langage ou canevas le plus approprié à la fonction métier de ce composant. Par exemple, WS-BPEL est adapté pour implanter des orchestrations de services avec compensation. Xquery est plus adapté pour de complexes traitements de données XML. Le langage C est plus adéquate pour écrire des pilotes de périphériques matériels. D'un autre coté, l'assembleur peut réutiliser des composants pré-existants sur l'étagère tels que des composites Fractal, des assemblages Spring, des bundles OSGi ou des artefacts EJB. **Une application s'exécutant sur FraSCAti est l'intégration de composants mettant en oeuvre des technologies d'exécution potentiellement hétérogènes**.

2.5.3.1 Le cas de l'intégration du langage WS-BPEL

Plutôt que de détailler l'intégration de chacun des langages et canevas supportés par FraSCAti [37], cette section détaille l'un des cas les plus techniquement intéressant : l'intégration du langage WS-BPEL.

La figure 2.7 présente l'implantation en cinq composants SCA d'un scénario de gestion de crise comme par exemple la gestion d'un accident routier. Le composant `Orchestration` encapsule le processus métier de gestion de la crise. Suite à une alerte reçue sur le service de ce composant, le processus métier va coordonner les activités de quatre acteurs : la police, les pompiers, la croix rouge et les services médicaux d'urgence. Chacun de ces acteurs est matérialisé par un composant SCA. Dans la réalité, chacun de ces composants serait hébergé par l'intervenant associé et serait accessible par une liaison sécurisée de type Web Services.

Pour réaliser une implantation WS-BPEL d'un composant SCA (ici le composant `Orchestration`), FraSCAti met en oeuvre le moteur d'exécution EasyBPEL de la société Linagora. Ce moteur EasyBPEL est lui-même implanté comme un assemblage de composants SCA (voir figure 2.7 - a). Cet assemblage est constitué de deux composants principaux : le compilateur WS-BPEL et le registre de processus WS-BPEL. Le compilateur WS-BPEL transforme un processus / fichier WS-BPEL en un assemblage de composants / composite SCA (figure 2.7 - b). Chaque instruction WS-BPEL est transformée en un composant SCA. Le registre de processus stocke et gère le cycle de vie des composites WS-BPEL. Pour réaliser l'intégration de EasyBPEL dans FraSCAti, j'ai réalisé un adaptateur qui convertit 1) les invocations de services SCA en messages XML qui sont envoyés au moteur EasyBPEL (figure 2.7 - c) et 2) les messages sortants du moteur EasyBPEL en soit la

37. Ce qui serait fastidieux et pas forcément très instructif.

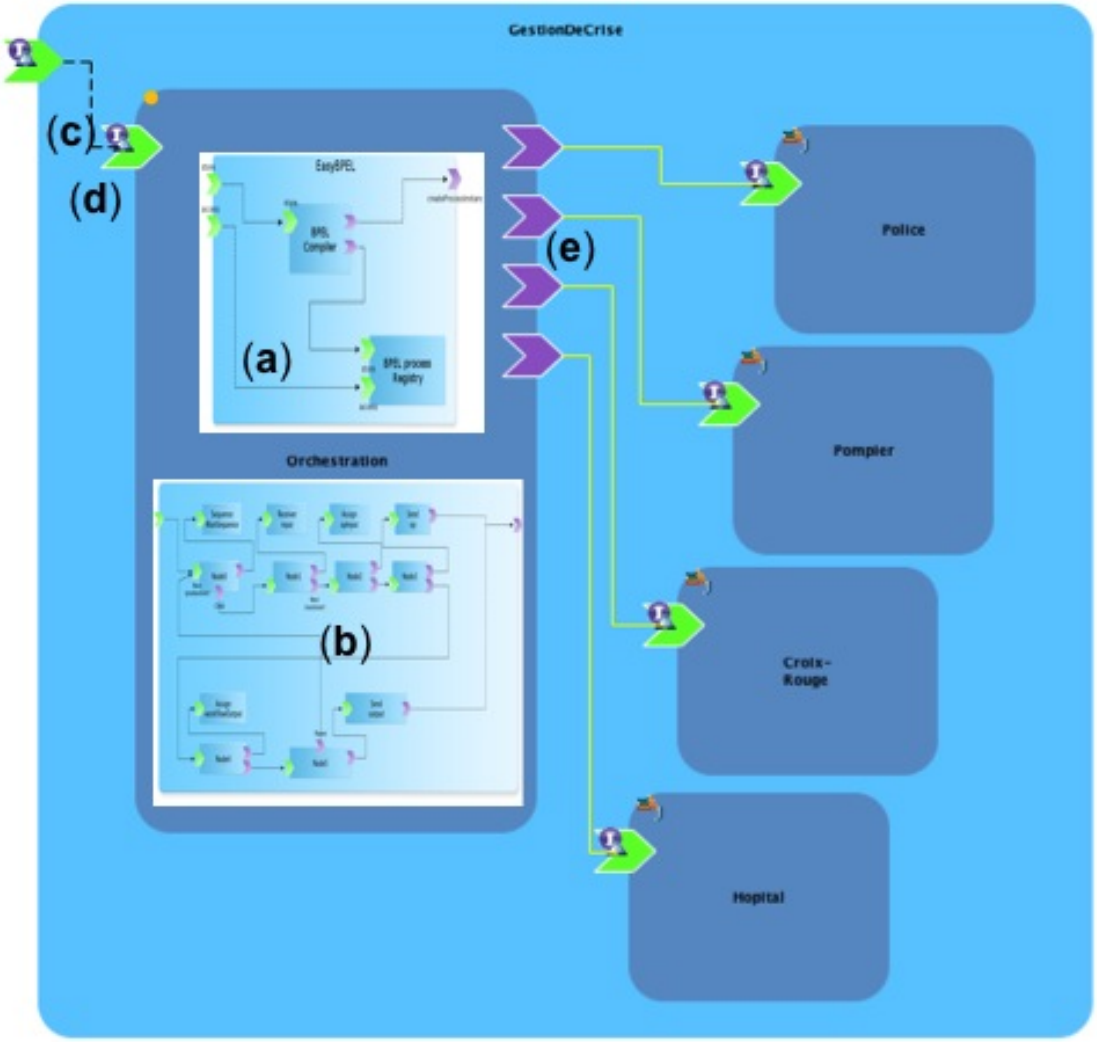

FIGURE 2.7 – L'intégration d'EASYBPEL avec FRASCATI.

réponse à une invocation de service (figure 2.7 - d) soit une invocation d'une référence du composant (figure 2.7 - e).

L'intégration de FraSCAti et EasyBPEL montre clairement que le modèle SCA et notre implantation FraSCAti peuvent être utilisés à différents étages d'une architecture ainsi qu'à différents niveaux de granularité d'intégration : bien sûr le composant métier mais aussi le moteur d'exécution d'une technologie d'implantation de composants puis chaque instruction d'un programme d'implantation. Ainsi, ce résultat conforte la thèse de Romain Rouvoy sur l'utilisation de composants à granularité extrêmement fine présentée précédemment dans la section 1.3.3.

Toutefois, nous pourrions nous demander quelle est le surcoût d'exécution et l'utilité de cette récursivité ? La société Linagora utilise EasyBPEL comme moteur d'exécution WS-BPEL pour son bus d'entreprises PEtALS qui est déployé chez de nombreux clients industriels. Linagora a évalué le coût d'exécution de EasyBPEL et celui-ci est similaire au coût d'exécution d'autres moteurs d'exécution WS-BPEL n'utilisant pas une approche à base de composants [38]. Côté utilité, la société Linagora a pu montré dans les projets ANR ARPEGE SALTY et SocEDA l'adaptabilité d'EasyBPEL et la reconfiguration à chaud de processus WS-BPEL parce que ceux-ci sont réalisés en composants SCA au dessus de FraSCAti. Au sein de l'équipe-projet Inria ADAM, la thèse de Gabriel Hermosillo [66] montre comment des processus métiers peuvent être dynamiquement adaptés en fonction du contexte. Pour valider son travail, Gabriel a mis en oeuvre le moteur EasyBPEL.

2.5.4 L'interopérabilité de multiples liaisons

Supporter différentes formes de liaisons d'interopérabilité apporte deux bénéfices complémentaires. Premièrement, une application SCA peut interopérer avec toute sorte d'applications non-SCA. Deuxièmement, cela permet de réaliser des orchestrations de services hétérogènes comme l'illustre la figure 2.2.

Conceptuellement, tout paradigme et intergiciel de communication peuvent être envisagés en termes d'interopérabilité. Concrètement, FraSCAti intègre des intergiciels d'appels synchrones à distance (RPC [39]), des intergiciels orientés messages (MOM [40]) et d'autres intergiciels comme OSGi et JGroups. En termes de RPC, FraSCAti supporte les courtiers de requêtes objets (ORB [41]) tel que Java RMI, les services Web tels que UPnP, les ressources Web (REST) tel que JSON RPC. Pour les services Web et les ressources REST, FraSCAti intègre la pile logicielle de services Apache CXF. FraSCAti propose aussi une liaison HTTP pour s'interfacer avec le World Wide Web (WWW). En termes de MOM, FraSCAti propose une liaison JMS basée sur l'intergiciel OW2 JORAM [42] et une liaison JBI basée sur l'intergiciel OW2 PEtALS. La liaison JGroups permet de réaliser des communications de groupes. La liaison OSGi permet de se connecter à un service OSGi pré-existant ou d'exposer un service SCA en tant que service OSGi. Cette liaison établit un pont bi-directionnel entre FraSCAti et l'annuaire de services d'une plate-forme OSGi.

38. Ces évaluations étant confidentielles, je n'ai pas de mesures à présenter dans ce manuscrit.
39. *Remote Procedure Call*
40. *Message Oriented Middleware*
41. *Object Request Broker*
42. `http://joram.ow2.org/`

2.5.4.1 Les composants de liaison

La réalisation des liaisons d'interopérabilité met en oeuvre un patron de conception indépendant des intergiciels d'interopérabilité supportés par la plate-forme FraSCAti. Ce patron que nous nommons *composant de liaison* est illustré par la figure 2.8. L'idée principal de ce patron est de réifier à l'exécution les liaisons SCA sous la forme de composants. Historiquement, cette idée fait écho aux *binding objects* proposés dans le modèle RM-ODP [218] et mis en oeuvre dans l'intergiciel Jonathan [41].

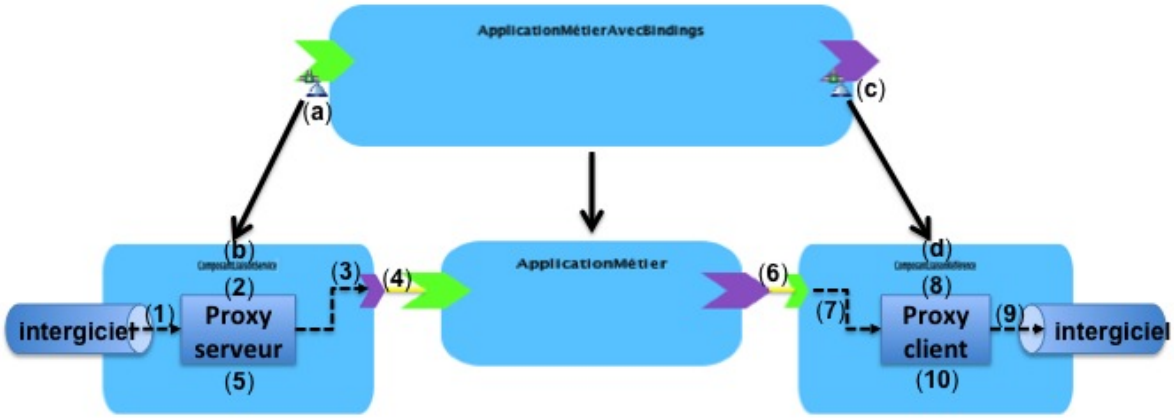

Figure 2.8 – Les composants de liaison d'interopérabilité.

Une liaison exportant un service SCA (figure 2.8 - a) est réifiée sous la forme d'un composant de liaison (figure 2.8 - b). Ce composant contient un *proxy serveur* pour l'intergiciel via lequel le service métier est exporté. Lorsque l'intergiciel doit délivrer une interaction (e.g., un appel de service, un message, etc.) alors l'intergiciel délivre cette interaction au proxy serveur (figure 2.8 - 1). Le rôle de ce proxy est de convertir l'interaction reçue en une invocation SCA (figure 2.8 - 2) et de l'envoyer à la référence du composant de liaison (figure 2.8 - 3). Cette référence est connectée au service métier exporté par la liaison (figure 2.8 - 4). Le résultat de l'invocation SCA est alors converti par le proxy serveur au format attendu par l'intergiciel de liaison (figure 2.8 - 5).

Une liaison associée à une référence SCA (figure 2.8 - c) est réifiée sous la forme d'un composant de liaison (figure 2.8 - d). Ce composant contient un *proxy client* pour l'intergiciel via lequel la référence doit être liée. La référence de l'application métier est connectée au service du composant de liaison (figure 2.8 - 6). Lorsque l'application métier invoque sa référence alors cette invocation est transmise jusqu'au proxy client (figure 2.8 - 7). Le rôle de ce proxy est de convertir les invocations SCA en interactions pour l'intergiciel utilisé (figure 2.8 - 8) puis de lui transmettre ces interactions (figure 2.8 - 9). En retour, le proxy client convertit le résultat de l'interaction en résultat d'invocation SCA (figure 2.8 - 10).

Ce patron *composant de liaison* est totalement générique et a été mis en oeuvre pour tous les types de liaisons supportés par FraSCAti (voir figure 2.6), c'est-à-dire les liaisons Java RMI, UPnP, Apache CXF Web Service et REST, JSON RPC, HTTP, JMS, JBI, OSGi et JGroups. Toutefois, les détails d'implantation des proxys serveurs et clients dépendent fortement de l'intergiciel utilisé et ne sont pas décrits dans ce manuscrit.

2.5.5 Les besoins extra-fonctionnels

Cette section discute de la mise en oeuvre des besoins extra-fonctionnels ou *intents* SCA présentés en section 2.2.2. Dans le contexte de FraSCAti, les besoins extra-fonctionnels sont réifiés sous la forme de composants extra-fonctionnels comme illustré par la figure 2.9. Ainsi les besoins extra-fonctionnels ont le même statut à la conception et à l'exécution que n'importe quelle application métier. Ces composants extra-fonctionnels sont tissés sur les composants, services et références des applications métiers. Sur la figure 2.9, le composant `Security` est tissé sur le service du composite métier. Le composant `Logging` est tissé sur tous les services et références du composant `View`. Le composant `Transaction` est tissé sur le composant métier `Model`.

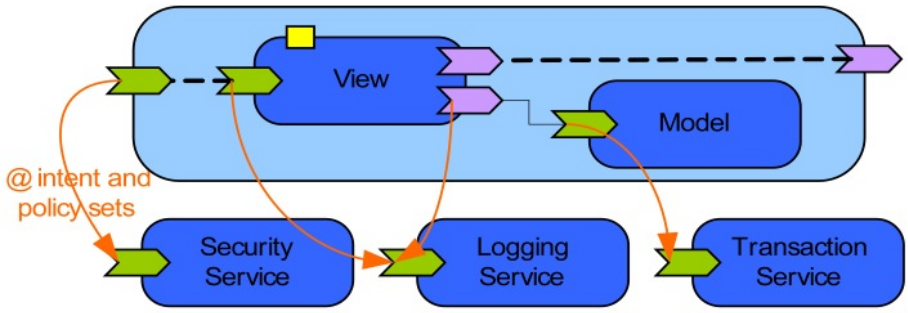

Figure 2.9 – Les composants extra-fonctionnels. Source : [214].

Les composants extra-fonctionnels reposent sur le patron de composant d'aspect présenté à la section 2.5.5.1. Ce patron est ensuite illustré par l'aspect de capture de diagramme de séquence UML en section 2.5.5.2, les aspects de liaisons Apache CXF en section 2.5.5.3 et l'aspect de sécurité dynamique en section 2.5.5.4.

2.5.5.1 Les composants d'aspect

Le patron de composant d'aspect FraSCAti est fortement inspiré de la thèse de Nicolas Pessemier [182] proposant l'unification des approches par aspects et à composants via le concept de composant d'aspect Fractal [184, 183, 185] . Une des idées de cette thèse est de matérialiser le greffon (*Advice*) d'un aspect via un composant. Ainsi le greffon devient une entité configurable via des propriétés au même titre que tout composant métier. De plus, un greffon complexe peut être réalisé par un assemblage de composants de complexité arbitraire.

La figure 2.10 présente la structure minimale d'un composant d'aspect FraSCAti. Celui-ci est un composite SCA constitué d'un composant gérant l'intention de l'aspect, c'est-à-dire la logique métier de l'*intent* SCA. Ce composant ainsi que le composite exposent un service de gestion d'intention. L'interface Java `IntentHandler` de ce service est une adaptation au monde des composants de l'interface proposée par l'AOP Alliance [43].

43. `http://aopalliance.sourceforge.net/`

FIGURE 2.10 – La structure minimale d'un composant d'aspect FRASCATI.

L'implantation minimale en JAVA d'un composant d'aspect est contenue dans le listing 2.2. Cette implantation doit implanter l'interface de gestion d'intention `IntentHandler` (ligne 1), soit l'unique méthode `invoke` (lignes 3 - 12). Cette méthode joue le rôle d'un intercepteur pouvant ajouter des traitements avant (ligne 5) et/ou après (ligne 9) l'exécution de la logique métier (ligne 7). Cet intercepteur peut éventuellement remplacer l'exécution métier par tout autre traitement. L'objet passé en paramètre permet d'obtenir l'ensemble des informations sur le point de jonction sur lequel l'intention a été posée (`IntentJoinPoint`). Cela comprend l'identité du composant invoqué, le service ou la référence invoqué, la méthode invoquée ainsi que la valeur effective des paramètres de cette méthode.

```
1  public class MyIntentHandler implements IntentHandler
2  {
3    public Object invoke(IntentJoinPoint ijp) throws Throwable
4    {
5      // Logique métier de l'intention à exécuter avant la logique métier.

7      Object ret = ijp.proceed();

9      // Logique de l'intention à exécuter après la logique métier.

11     return ret;
12   }
13 }
```

Listing 2.2 – L'implantation minimale en JAVA d'un composant d'aspect

Ainsi, nous pouvons constater qu'il est assez aisé de concevoir et réaliser de nouvelles propriétés extra-fonctionnelles. Celles-ci peuvent être généralistes (observation, sécurité, transaction, etc.) ou spécifiques à un besoin métier donné. Dans le cadre de l'ADT Inria GALAXY, nous avons développé un aspect SCA de disponibilité. Cet aspect interrompt l'invocation de service en cours au bout d'un laps de temps donné si aucune réponse n'a été renvoyée. Dans le cadre de sa thèse [43], Alexandre Feugas a développé un *intent* pour mesurer les temps d'exécution de processus métiers. Le projet SOFA a développé des *intents* FRASCATI afin de détecter dynamiquement des anti-patterns dans des architectures SCA[140, 170].

Les sections suivantes présentent trois exemples d'aspects complexes.

2.5.5.2 La capture de diagramme de séquence UML

L'objectif de ce composant d'aspect est de capturer les traces d'exécution d'applications métiers puis d'en générer une représentation graphique sous la forme de diagrammes de séquences UML. En bref, ce composant d'aspect propose une forme d'introspection comportementale des applications. Cela peut s'avérer très utile lors du déboggage d'applications orientées services mais aussi cela permet d'envisager de vérifier que le comportement des composants (ou au moins leurs interactions) est conforme avec les diagrammes de séquences réalisés à la conception d'une application.

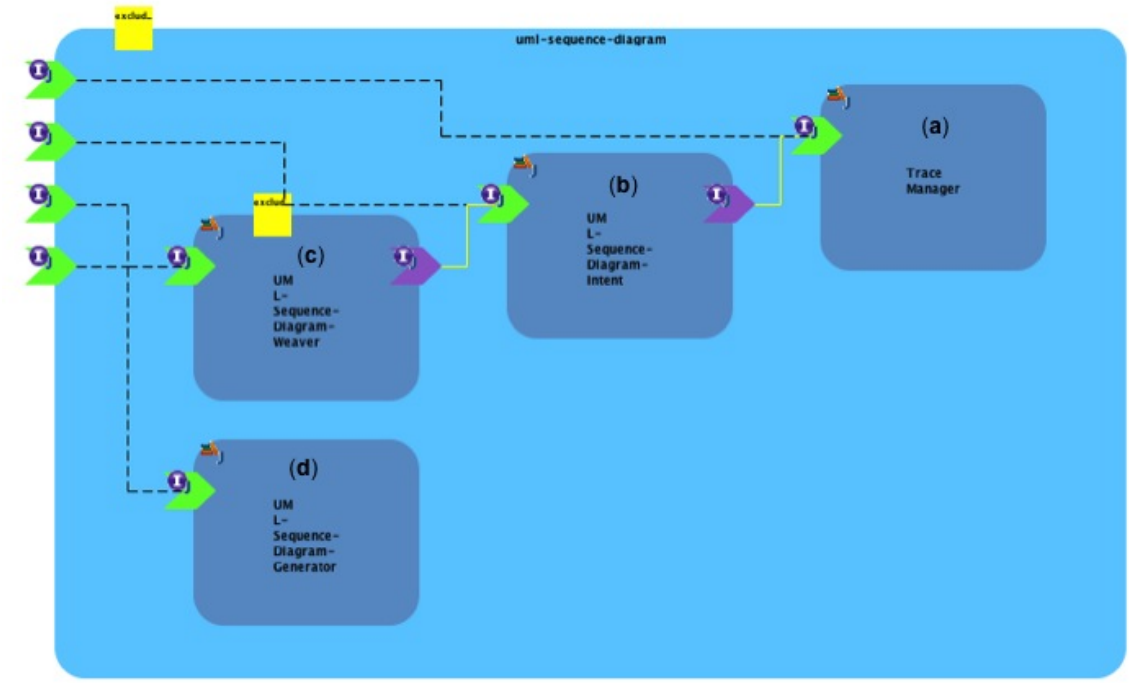

FIGURE 2.11 – L'aspect FRASCATI de capture de diagrammes de séquences UML.

Cet exemple illustre la construction d'un composant d'aspect complexe. Cet aspect est la composition de quatre composants comme l'illustre la figure 2.11 :

1. le composant (a) stocke les traces d'exécution collectées.
2. le composant (b) est le greffon de collecte de traces d'exécution.
3. le composant (c) tisse / détisse dynamiquement le greffon (b) sur la clôture transitive des composants d'une application métier.
4. le composant (d) traduit les traces d'exécution en diagrammes de séquences UML tel que celui-ci de la figure 2.12.

Cet exemple montre qu'un composant d'aspect n'est pas seulement constitué d'un greffon (b) mais peut aussi contenir la logique de tissage (l'expression des *point cuts* dans la terminologie aspect) ainsi qu'une logique métier, le tout accessible via des services métiers. De plus, cet aspect peut tout à fait être tissé sur d'autres aspects pour capturer les interactions de ceux-ci.

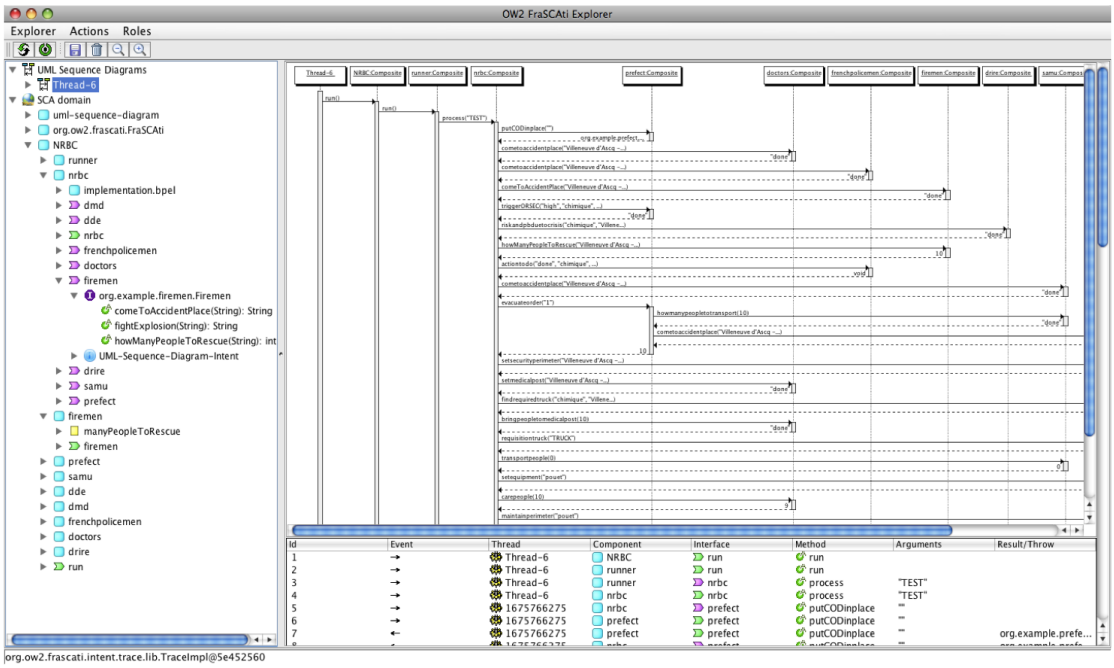

FIGURE 2.12 – Un exemple de capture de diagramme de séquence UML.

Comme perspective, cet aspect pourrait être étendu pour capturer des interactions entre composants répartis en s'inspirant de travaux portant sur les objets répartis tels que GOODEWATCH [59] ou [40]. Cette extension nécessiterait de modifier le comportement des liaisons d'interopérabilité via des aspects tel que présenté dans la section suivante.

2.5.5.3 Les aspects de liaisons Apache CXF

FRASCATI met en oeuvre le cadriciel APACHE CXF pour réaliser les liaisons d'interopérabilité Web Service et REST. APACHE CXF supporte de nombreux standards web services incluant le protocole de communication SOAP [231, 232, 233], le profil d'interopérabilité (*WS-I Basic Profile* [241]), le langage de description de services WDSL [235, 236, 237], WS-ADDRESSING [229, 230, 234], WS-DISCOVERY [146], WS-POLICY [238], WS-RELIABLEMESSAGING [144], WS-SECURITY [143], WS-SECURITYPOLICY [157], WS-SECURECONVERSATION [147], WS-TRUST [158] ainsi que de nombreuses fonctionnalités non standards pour configurer les services web. Afin de répondre aux besoins des partenaires industriels du projet FUI EASYSOA [44], j'ai réalisé une bibliothèque de composants d'aspects FRASCATI pour configurer sur les liaisons APACHE CXF. Cette bibliothèque fournit en autres des aspects pour fixer le délai (*time out*) de connexion à un service distant ou le délai de réception de messages, utiliser un serveur proxy intermédiaire pour accéder à des services distants, gérer la redirection automatique des messages lorsqu'un service a changé de localisation, crypter les messages via TLS/SSL, authentifier les échanges, compresser les messages avec FASTINFOSET ou GZIP, gérer les longs messages XML en mode streaming, modifier le contenu des messages, choisir la technologie d'emballage/déballage des données, utiliser le standard WS-ADDRESSING, fiabiliser les échanges avec WS-RELIABLEMESSAGING,

44. `http://www.easysoa.org/`

mettre en place un pare-feu dédié à chaque service, découvrir les services avec le standard WS-Discovery, regrouper des services pour la gestion des pannes (*failover*) et la répartition de charge. Ces différents aspects peuvent être combinés comme illustré dans le scénario suivant.

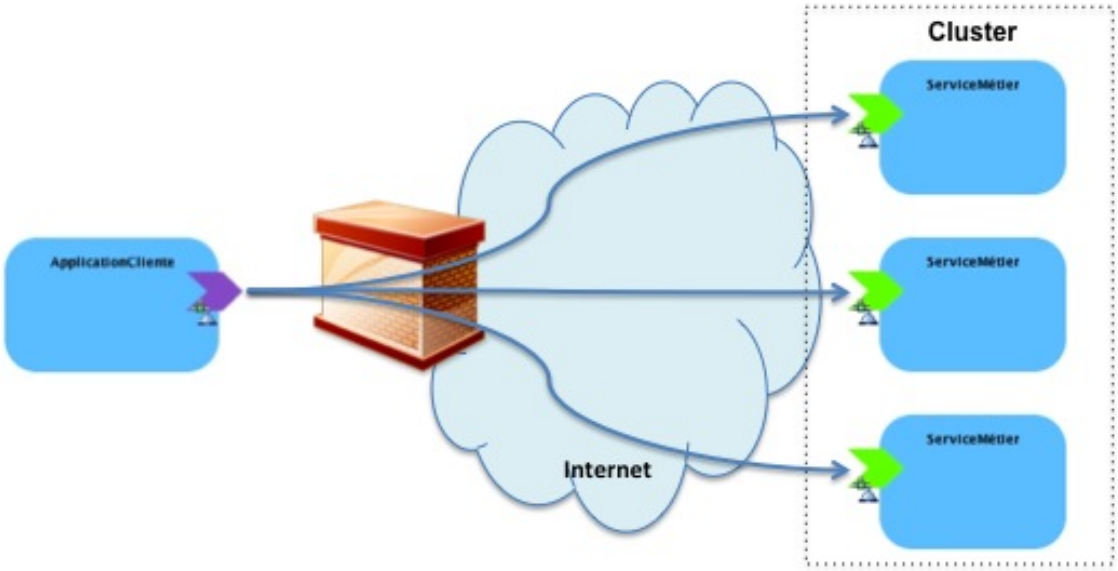

Figure 2.13 – Un scénario de répartition de la charge sur un cluster de services Web.

La figure 2.13 illustre un scénario de répartition de la charge au sein d'un cluster de services Web. L'application cliente utilise un service Web disponible sur Internet. La découverte du service est effectué via le protocole WS-Discovery. Comme l'application cliente est sur un réseau local, ses interactions avec le service doivent passer à travers un pare-feu. Il est nécessaire de fixer les délais de connexions et de réception de messages pour éviter d'attendre un laps de temps trop grand lorsque le réseau est congestionné ou que le service est indisponible ou surchargé. Les messages échangés sont compressés pour réduire le trafic réseau. Les échanges de messages sont fiabilisés via le protocole WS-ReliableMessaging. Le service a besoin d'authentifier ses clients pour des besoins de facturation. Ce service est réalisé par un groupe de répliquas afin de répartir la charge de travail et gérer les pannes du service.

L'ensemble de ces besoins extra-fonctionnels est exprimé via des intentions sur la liaison Web service de l'application cliente comme illustré dans le listing 2.3 en lignes 4 et 5. Certains de ces besoins nécessitent d'être paramétrés : l'adresse du pare-feu (lignes 11 et 12), la nature de la technique de compression des messages (ligne 15), le seuil à partir duquel les messages sont compressés (ligne 16), le délai maximal lors de la connexion au service (ligne 20), le délai maximal d'attente des réponses du service (ligne 24) ainsi que la politique d'authentification et les informations associées (lignes 27 à 29). Pour ces besoins, les composants d'aspect sont déclarés dans l'assemblage de l'application cliente et sont donc des aspects locaux, non partagés et dédiés à celle-ci. Les autres intentions ne nécessitent pas de paramètres (`WS-ReliableMessaging` et `WS-Discovery-LoadBalancing-RandomStrategy`) et sont donc instanciées à l'extérieur de l'application cliente comme illustré dans la figure 2.14. Ce sont donc des aspects partagés. Ces derniers peuvent alors être partagés entre plusieurs composites métiers s'exécutant sur le même noeud. Le composant d'aspect `WS-Discovery-`

`LoadBalancing-RandomStrategy` encapsule une politique extra-fonctionnelle complexe : 1) la liste des adresses du service est obtenue via le protocole `WS-Discovering`, 2) chaque invocation du service est dirigée vers une des adresses de la liste afin d'assurer une répartition de la charge et 3) l'adresse du service est tirée au hasard de la liste.

```
1  <composite name="ApplicationCliente">
2    <reference name="r" promote="logique-métier/r">
3      <binding.ws ...
4        require="pare-feu compression délai-connexion délai-réception authentification
5                 WS-ReliableMessaging WS-Discovery-LoadBalancing-RandomStrategy"/>
6    </reference>
7    <component name="logique-métier"> <reference name="r"/> ... </component>
8    <component name="pare-feu">
9      <implementation.composite name="ProxyServerIntent"/>
10     <property name="ProxyServer">217.108.22.220</property>
11     <property name="ProxyServerPort">80</property>
12   </component>
13   <component name="compression">
14     <implementation.composite name="GZIPIntent"/>
15     <property name="Threshold">1024</property>
16   </component>
17   <component name="délai-connexion">
18     <implementation.composite name="ConnectionTimeoutIntent"/>
19     <property name="ConnectionTimeout">10000</property>
20   </component>
21   <component name="délai-réception">
22     <implementation.composite name="ReceiveTimeoutIntent"/>
23     <property name="ReceiveTimeout">60000</property>
24   </component>
25   <component name="authentification">
26     <implementation.composite name="BasicAuthorizationPolicyIntent"/>
27     <property name="UserName">merle</property>
28     <property name="Password">philippe</property>
29   </component>
30 </composite>
```

Listing 2.3 – Le composite de l'application cliente d'un cluster de services Web

La figure 2.14 montre le tissage des sept composants d'aspect sur la liaison Web service qui est elle-même réifié sous la forme d'un composant de liaison, voir la section 2.5.4.1.

Cet exemple illustre clairement que **FraSCAti propose un modèle de construction d'applications orientées services unifiant l'expression des fonctionnalités métiers et l'expression des propriétés extra-fonctionnelles au travers de l'unique paradigme des composants**.

Pour aller plus loin sur cet exemple, il serait nécessaire de gérer la synchronisation des états des composants répliquées du cluster. Cela a été effectuée dans la thèse de Miruna Stoicescu [220] dans laquelle une grande variété de mécanismes de tolérances aux pannes (dont la gestion des états) a été implantée par des composants SCA s'exécutant au dessus de FraSCAti.

2.5.5.4 Model Driven Security@run.time

Depuis janvier 2014, j'ai entamé une collaboration avec la Prof. Frédérique Bernier et le docteur Wendpanga Francis Ouedraogo, tous deux membres du LIRIS à Villeurbanne, sur la sécurisation de services déployés dans les nuages. Dans ce contexte ouvert, des utilisateurs apparaissent et disparaissent durant l'exécution des services. Ainsi les politiques de sécurité des services doivent être modifiées dynamiquement. De plus, les politiques de sécurité peuvent être optimisées en fonction du contexte. Par exemple, il n'est pas nécessaire d'encrypter les

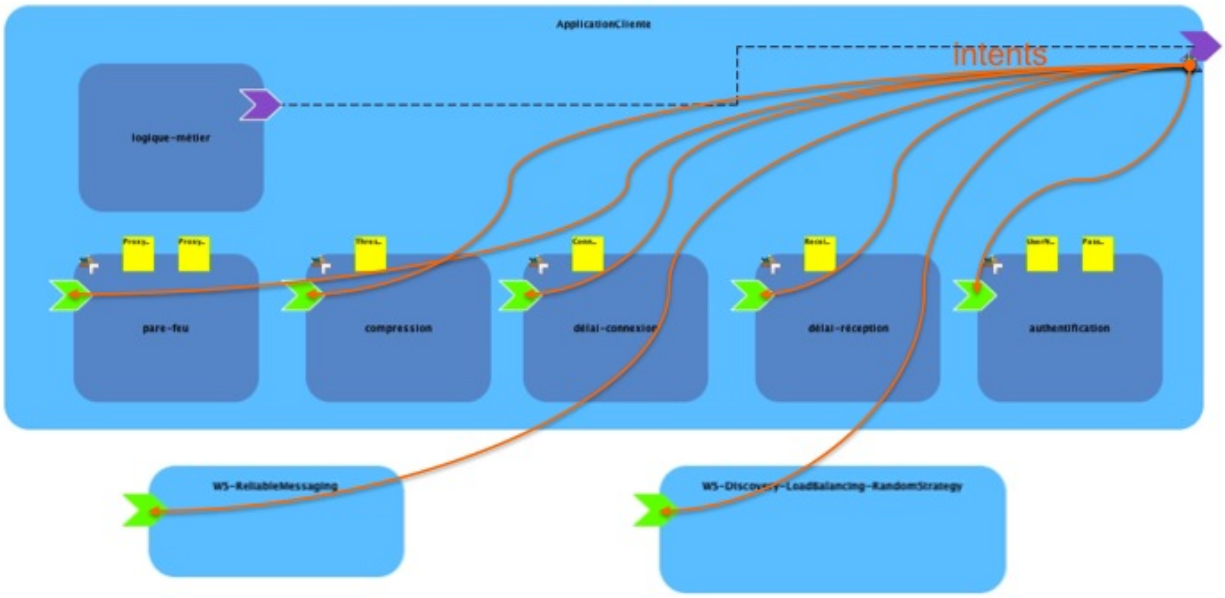

FIGURE 2.14 – La représentation graphique du composite du listing 2.3.

messages applicatifs si le réseau sous-jacent est un réseau virtuel privé encryptant déjà les messages échangés. Ainsi nous pouvons éviter de surprotéger un système distribué. Pour cela, nous travaillons sur une sécurisation dynamique, adaptable et contextuelle des services déployés dans les nuages et avons proposé l'approche novatrice MODEL DRIVEN SECURITY@RUN.TIME [164, 163]. Cette approche combine deux approches bien connues, à savoir la sécurité dirigée par les modèles (MODEL DRIVEN SECURITY ou MDS en abrégé) [9, 8, 88] et l'approche MODELS@RUN.TIME [14]. L'approche MDS prône une ingénierie dirigée par les modèles pour la conception et la vérification des politiques de sécurité. Ces politiques sont alors matérialisées par des modèles. L'approche *Models@run.time* prône de piloter le comportement et l'adaptation dynamique de systèmes logiciels via des modèles présents à l'exécution. Notre combinaison prône donc de matérialiser les politiques de sécurité sous la forme de modèles et de mettre en oeuvre la sécurité par interprétation à l'exécution de ces modèles de politiques de sécurité.

Nous avons mis en oeuvre un prototype de MDS@RUN.TIME dont l'architecture à composants FRASCATI est illustrée par la figure 2.15. Ce prototype se compose de trois composites :

- **Interceptor** est un composant d'aspects FRASCATI qui interceptent les invocations des services transportées par l'intergiciel APACHE CXF. Ces invocations sont alors réifiées et transmises à l'interprète MDS@RUN.TIME.
- **MDS@run.time** contient l'interpréteur des modèles de politiques de sécurité (composant `Mediator`). Celles-ci sont obtenues auprès d'un dépôt de politiques de sécurité (composant `Policy Manager`). Ces politiques sont contextualisées via le gestionnaire de contexte (composant `Context Manager`). La réalisation effective de la sécurité est déléguée au composite `SecaaS`.
- **SecaaS** pour SECURITY AS A SERVICE contient divers composants encapsulant les fonctions de sécurité telles que l'authentification, l'autorisation, l'encryptage des données, le

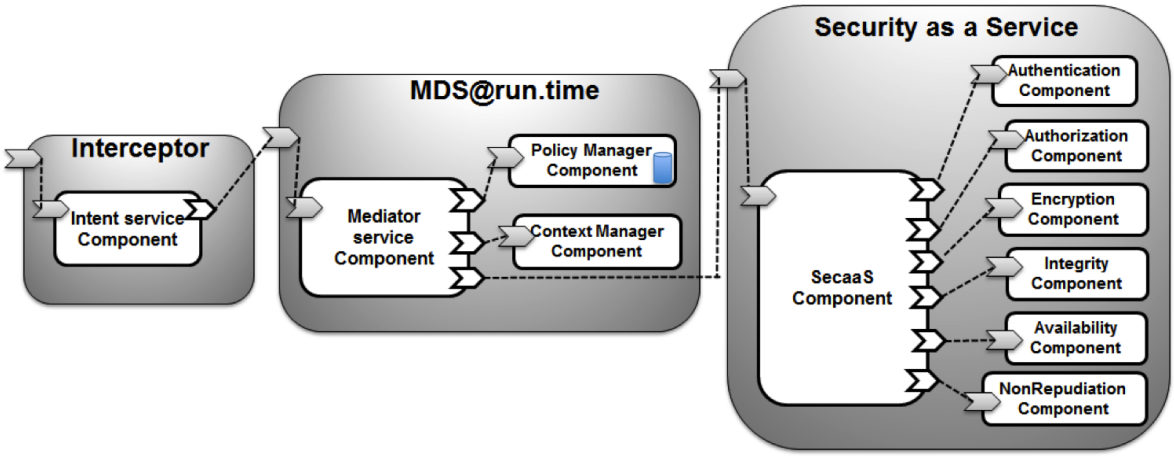

FIGURE 2.15 – L'architecture de MDS@RUN.TIME. Source : [163].

contrôle de l'intégrité, la gestion de la disponibilité et de la non-répudiation.

Le prototype MDS@RUN.TIME a été évalué sur plusieurs scénarios applicatifs. Le tableau 2.1 présente l'évaluation des temps moyens d'exécution pour le scénario présenté dans le papier [163]. Le temps moyen d'invocation du service métier sans sécurisation ni MDS@RUN.TIME (1) est de 53 millisecondes soit 75% du temps total d'exécution observé quand MDS@RUN.TIME est mis en place. La mise en oeuvre de deux fonctions de sécurité - authentification (4) et autorisation (5) - coûte de l'ordre de 17% du temps moyen total. Nous pouvons ainsi observer que le temps moyen de l'interception FRASCATI des échanges APACHE CXF (2) et de l'interprétation contextuelle des modèles de politiques de sécurité (3) n'est pas négligeable mais reste très faible (respectivement 1 et 4 millisecondes, soit 2% et 6% du temps total) par rapport à l'exécution du service métier et des services de sécurité. Ainsi ce prototype démontre que notre approche d'interprétation dynamique et contextuelle des politiques de sécurité n'engendre pas un surcoût important (de l'ordre de 8%).

TABLE 2.1 – Les temps moyens d'exécution des composantes de MDS@RUN.TIME.

No	Composant	Temps moyen (ms)	Temps moyen / Temps total
1	FRASCATI + APACHE CXF + service métier	53	75%
2	FRASCATI `Interceptor`	1	2%
3	`MDS@RUN.TIME`	4	6%
4	`Authentication`	10	14%
5	`Authorization`	2	3%
	Total	70	100%

Sur des scénarios plus complexes, c'est-à-dire mettant en oeuvre plus de deux fonctions de sécurité, nous escomptons que l'interprétation dynamique des politiques de sécurité permettra d'éviter la sur-protection et ainsi de réduire les temps d'exécution de services sécurisés. Dans [165], nous avons déjà montré que MDS@RUN.TIME permet de réduire les temps d'exécution lorsque le risque de sur-protection est supérieur ou égal à 30%. La seconde perspective de ce travail est d'expérimenter la modification à chaud des modèles de politiques de sécurité

stockés dans le composant `Policy Manager`.

2.5.6 La plate-forme FraSCAti

La plate-forme FraSCAti est constituée de deux parties principales comme illustré par la figure 2.16 : l'interpréteur de descripteurs SCA et le support d'exécution FraSCAti. L'interpréteur FraSCAti analyse les descripteurs SCA fournis en entrée puis instancie les assemblages de composants SCA associés. Le support d'exécution FraSCAti fournit les fonctionnalités requises pour l'exécution et la reconfigurabilité des composants SCA et est discuté plus en détails dans la section 2.7.1.

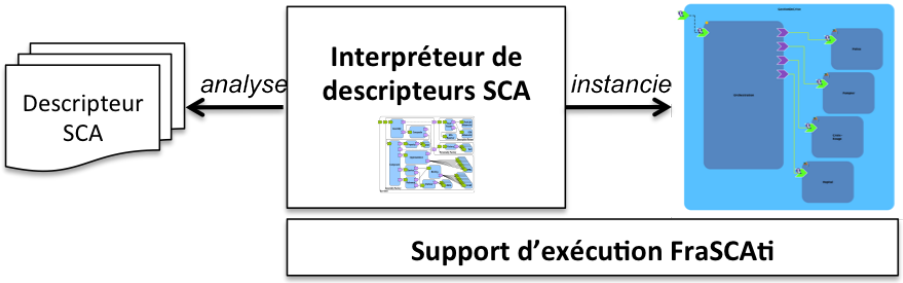

Figure 2.16 – La plate-forme FraSCAti.

L'interpréteur FraSCAti est lui-même un assemblage de composants SCA qui réalise la flexibilité, l'extensibilité et l'adaptabilité de l'intergiciel d'intergiciels FraSCAti (voir figure 2.6). Cet assemblage est constitué de trois composites comme l'illustre la figure 2.17 :

- **Description Parser** : le parseur de descriptions a pour rôle de charger les descripteurs SCA puis de les réifier en mémoire sous la forme d'un arbre de syntaxe abstraite (une structure de données dont les noeuds typés contiennent les informations stockées dans les balises XML du langage de descripteurs SCA). Pour cela, nous avons choisi d'utiliser la technologie Eclipse Modeling Framework (EMF) [45] [219]. Le langage SCA est pris en charge par le méta-modèle SCA fourni par le projet Eclipse SCA Tools. Ce choix nous permet d'assurer un continuum sans couture entre les outils de modélisation Eclipse pour SCA et notre plate-forme FraSCAti. Chaque extension du langage SCA est encodée dans un méta-modèle distinct comme par exemple le méta-modèle pour les extensions SCA spécifiques à FraSCAti.
- **Assembly Factory** : la fabrique d'assemblage a pour rôle de vérifier la cohérence sémantique des arbres de syntaxe abstraites produits par le parseur puis d'instancier les structures d'exécution associées. Pour cela, chaque construction du langage SCA, comme par exemple `<composite>`, `<component>`, `<implementation>`, `<binding.ws>`, etc., est géré par un composant dédié, désigné sous le vocable de processeur FraSCAti. Ces processeurs sont assemblés entre eux selon l'imbrication des constructions du langage SCA, par exemple la construction `<composite>` contient des `<component>` qui ont une `<implementation>`, etc. Alors le processeur `Composite` est connecté au processeur `Component` qui est relié au processeur `Implementation`. Les processeurs collaborent par l'intermédiaire d'une structure de données partagées nommée le contexte de traitement (`ProcessingContext`). Chaque processeur offre au moins un service dont l'interface

45. `http://www.eclipse.org/modeling/emf/`

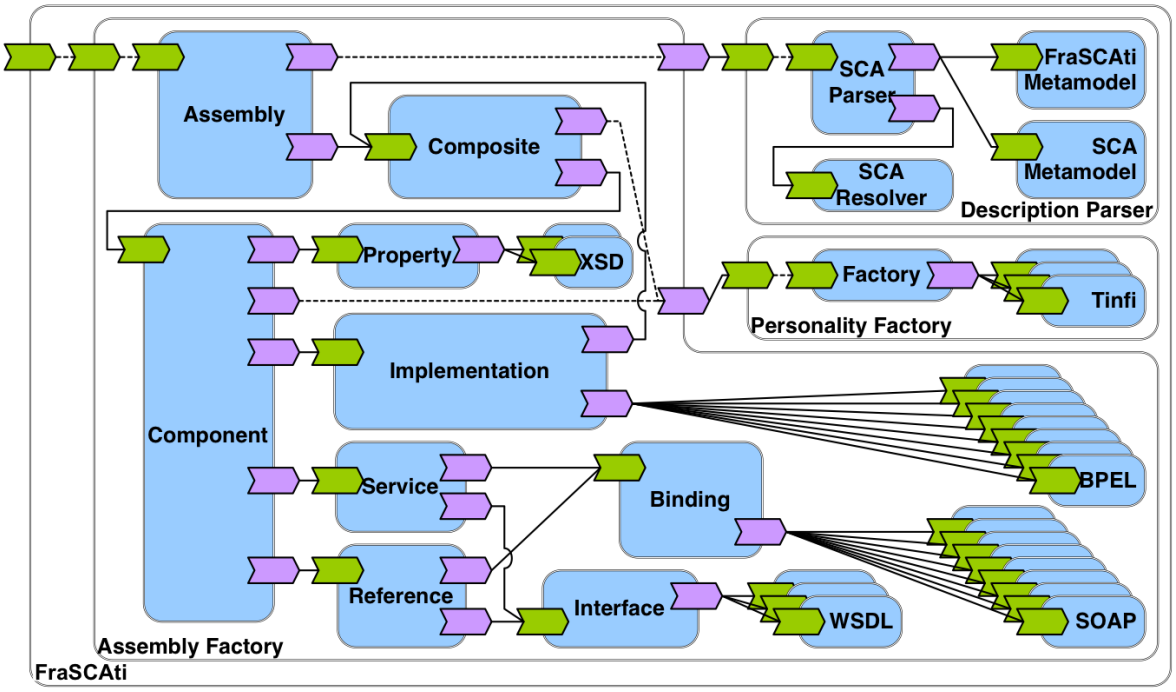

FIGURE 2.17 – L'architecture simplifiée de l'interpréteur FRASCATI de descripteurs SCA.

JAVA est présentée dans le listing 2.4. Un processeur a une identité unique (méthode `getProcessorId`) et fournit quatre méthodes de traitement. La méthode `check` a pour but de vérifier sémantiquement la construction SCA passée en paramètre. Par exemple pour la construction `<service>`, cette méthode vérifie en autres que le nom du service est unique au sein du composant parent. La méthode `generate` a pour but de générer les artefacts associés à la construction SCA. Par exemple pour la construction `<interface.wsdl>`, cette méthode compile l'interface WSDL afin de générer le code JAVA associé à cette interface. La méthode `instantiate` a pour but d'instancier les structures d'exécution associées à la construction SCA. Par exemple pour la construction `<implementation.bpel>`, cette méthode invoque le moteur EASYBPEL afin de charger le processus WS-BPEL et de créer le composite SCA réifiant ce processus (voir la section 2.5.3.1). Pour les constructions `<binding.*>`, cette méthode instancie le patron *composant de liaison* présentée dans la section 2.5.4.1. La méthode `complete` a pour but de finaliser les structures d'exécution associées à la construction SCA. Par exemple pour la construction `<composite>`, cette méthode démarre le composite.

Ajouter une nouvelle construction au langage SCA nécessite alors d'ajouter le nouveau processeur prenant en charge la sémantique de cette construction, ce qui revient concrètement à implanter les cinq méthodes de l'interface `Processor`.

- **Personality Factory** : cette fabrique a pour rôle d'instancier concrètement la structure d'exécution pour les composites et composants SCA. Selon la nature de l'implantation d'un composant SCA, cette fabrique utilise une personnalité dédiée. `Tinfi` est la personnalité pour les composants implantés en JAVA et est détaillée dans la section 2.7.1.

```
1 @org.oasisopen.sca.annotation.Service
```

```
2 public interface Processor<ScaConstruct>
3 {
4   String getProcessorID();
5   void check      (ScaConstruct c, ProcessingContext pc) throws ProcessingException;
6   void generate   (ScaConstruct c, ProcessingContext pc) throws ProcessingException;
7   void instantiate(SCAConstruct c, ProcessingContext pc) throws ProcessingException;
8   void complete   (SCAConstruct c, ProcessingContext pc) throws ProcessingException;
9 }
```

Listing 2.4 – L'interface JAVA d'un processeur FRASCATI

Le canevas SOFA/SODA permettant de détecter les anti-patterns SOA a été appliqué sur l'architecture de FRASCATI [46]. Dans [170], leur étude conclut que :

> *" In summary, very few services, i.e., 10 are actually involved in 6 antipatterns (4 antipatterns are not present) in FraSCAti, in comparison to the high number of services, i.e., 130 services. FraSCAti is well designed with continuous maintenance and evolution. Mostly, services (e.g., sca-parser and sca-composite-*) related to parsing and handling the composite file are involved in the antipatterns. The presence of such antipatterns in a system is not surprising because there is no other way to develop a parser without introducing a high coupling among services. "*

Plus précisément concernant l'interpréteur FRASCATI de descripteurs SCA, seulement le composant `SCA Parser` implante trois anti-patterns : *Tiny Service* car il n'offre qu'un seul service avec une seule méthode `parse`, *The Knot* et *Bottleneck Service* car c'est le seul composant en charge de lire les descripteurs SCA depuis le système de fichiers (opération I/O généralement coûteuse en temps d'exécution). Ainsi nous pouvons conclure comme les auteurs de [170] que FRASCATI est bien conçu.

Comme nous venons de le voir, la plate-forme FRASCATI fournit un interpréteur du langage SCA. Cet interpréteur se base sur une architecture à composants réflexive et extensible. Chaque construction du langage SCA est réifiée par un composant **processeur**. L'ajout de nouveaux processeurs permet de prendre en compte de nouvelles constructions.

Il me semble que cette approche pourrait être généralisée par un patron de conception orienté composant pour la construction de l'interpréteur d'un langage quelconque (pas uniquement le langage SCA, ni des langages XML). A partir d'une définition du langage (une grammaire BNF, un schéma XML, etc.), il devrait être possible de générer le squelette de l'architecture de l'interpréteur extensible de ce langage ainsi que le squelette du code d'implantation des processeurs pour ce langage, réduisant ainsi grandement le temps de développement et améliorant sensiblement la fiabilité de l'interpréteur résultant.

2.5.7 Synthèse sur QR1

Pour résumer, les sections précédentes ont montré à travers de nombreux exemples que la plate-forme FRASCATI repose sur deux idées extrêmement simples mais puissantes : **un intergiciel d'intergiciels et des composants partout**.

Nous avons montré que le paradigme composant peut être systématiquement appliqué à tous les niveaux d'un système logiciel distribué :

- l'architecture des applications métiers,
- le moteur d'exécution d'une technologie d'implantation, e.g., le moteur EASYBPEL,

46. `http://sofa.uqam.ca/soda/`

- les instructions d'une implantation, e.g., un processus WS-BPEL réifié en un composite SCA,
- les liaisons d'interopérabilité,
- les propriétés extra-fonctionnelles aussi bien des composants que des liaisons,
- l'intergiciel d'intergiciels lui-même, e.g., FraSCAti.

Cela répond ainsi à notre première question de recherche (QR1), à savoir comment supporter la flexibilité / extensibilité / adaptabilité d'un modèle orienté service à base de composants logiciels, aka SCA.

2.6 QR2 : Modulariser la flexibilité / extensibilité / adaptabilité d'une plate-forme SCA

Cette section présente notre réponse à la deuxième question de recherche (QR2) : comment modulariser la flexibilité / extensibilité / adaptabilité d'une plate-forme SCA ?

Bien qu'une plate-forme SCA doive supporter différentes technologies d'implantation des composants et des liaisons, différents langages de programmation et de description d'interfaces, diverses propriétés extra-fonctionnelles, **une même application métier nécessite rarement l'ensemble de ces technologies simultanément**.

Par exemple, toutes les applications n'ont pas besoin d'un moteur d'exécution WS-BPEL, seules celles qui nécessitent des mécanismes de compensation pourront contenir des implantations en WS-BPEL. Seules les applications manipulant de gros volumes de données ont intérêt à utiliser le langage Xquery. Toutes les applications ne nécessitent pas des liaisons d'interopérabilité de type services Web ou REST, ainsi toute application SCA ne doit pas embarquer une pile logiciel orientée service tel que Apache CXF. De même, une application embarquée limitera les technologies utilisées afin de réduire son empreinte mémoire, disque et/ou CPU.

Ainsi, il convient de **proposer une plate-forme intergicielle modulable " à la carte "**, c'est-à-dire que les fonctionnalités réellement présentes à l'exécution doivent pouvoir être sélectionnées en fonction des besoins réels des applications à exécuter. Historiquement, cette piste a déjà été explorée comme par exemple dans JonasALaCarte, un serveur d'applications J2EE administrable, proposé dans la thèse de Takoua Abdellatif [1].

Comme discuté dans la section 2.5.6, l'architecture de FraSCAti est un assemblage de composants SCA proposant trois points de flexibilité / extensibilité / adaptabilité : les composants **méta-modèle** du composite `Description Parser` (e.g., méta-modèle FraSCAti), les composants **processeur** du composite `Assembly Factory` (e.g., processeur WS-BPEL) et les composants **personnalité** (e.g., composant `Tinfi`). Ces composants sont désignés par la suite sous le vocable de **composants d'extension**. Chacun de ces composants réalise l'intégration d'une technologie au sein de la plate-forme FraSCAti. Les dépendances entre la plate-forme FraSCAti et une technologie donnée sont circonscrites (encapsulées et limitées) à un composant dans la plupart des cas. Ainsi il faudrait pouvoir choisir à la demande quels composants seront inclus dans la description d'architecture de la plate-forme FraSCAti, c'est-à-dire dans les composites `FraSCAti`, `Description Parser`, `Assembly Factory` et `Personality Factory` de la figure 2.17.

Notre réponse pour modulariser la flexibilité / extensibilité / adaptabilité d'une plate-forme SCA repose sur deux idées extrêmement simples mais puissantes : une plate-forme modulable " à la carte " et la fragmentation de la description de l'architecture de la plate-forme.

2.6.1 Une plate-forme modulable " à la carte "

La plate-forme FraSCAti est décomposée en plus d'une cinquantaine de modules comme l'illustre la figure 2.18. Chaque module est implanté sous la forme d'un module Maven produisant un artefact déployable Java ARchive (JAR).

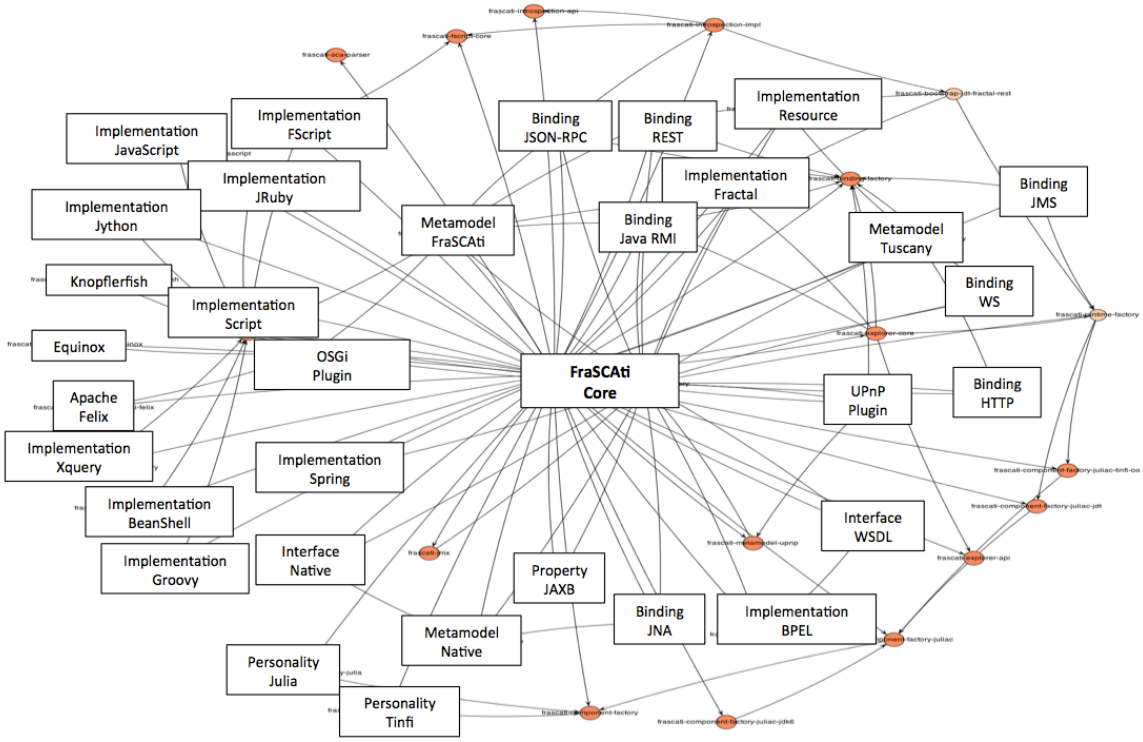

Figure 2.18 – Les modules de FraSCAti et le graphe de leurs dépendances.

Le module principal nommé `FraSCAti Core` contient l'architecture de base de la plate-forme FraSCAti incluant un minimum de composants d'extension, à savoir les composants `SCA Metamodel`, `Tinfi`, `Property Java`, `Implementation Java`, `Binding SCA` et `Interface Java`. Chacun des autres composants d'extension est contenu dans un module spécifique comme les méta-modèles `Metamodel FraSCAti`, `Metamodel Tuscany`[47] et `Metamodel Native`[48], les processeurs pour `Property XSD`[49], `Implementation BPEL`, `Implementation Resource`[50], `Implementation Spring`, `Implementation Fractal`, les langages interprétés `Im-`

47. Ce méta-modèle permet de traiter les extensions au langage SCA proposées par la plate-forme Apache Tuscany.

48. Ce méta-modèle décrit nos extensions au langage SCA pour désigner des fichiers *headers* en langage C et des bibliothèques natives.

49. pour les valeurs de propriétés exprimées avec le langage XML.

50. Ce type d'implantation gère un ensemble de ressources Web statiques tels que des pages HTML, des

plementation FScript, Implementation JavaScript, Implementation JRuby, Implementation Jython, Implementation Xquery, Implementation BeanShell, Implementation Groovy, les liaisons d'interopérabilité Binding Java RMI, Binding WS, Binding JMS, Binding REST, Binding JSON-RPC, Binding HTTP, Binding JNA[51], les interfaces Interface WSDL et Interface Native. Les modules OSGi Plugin et UPnP Plugin contiennent chacun plusieurs composants d'extension nécessaires pour intégrer respectivement les technologies OSGi et UPnP au sein de la plate-forme FraSCAti.

2.6.2 La fragmentation de la description de l'architecture de la plate-forme

Chaque module contient un ou plusieurs composants constituant la plate-forme FraSCAti. Ainsi nous pouvons considérer que la description de l'architecture de la plate-forme FraSCAti est fragmentée dans les différents modules.

Par exemple, le module FraSCAti Core contient le fragment principal du composite Assembly Factory (voir le listing 2.5). Ce fragment contient entre autres les processeurs Implementation (lignes 2 à 5) et Interface (lignes 6 à 9). Chacun de ces composants possède une référence multiple sur laquelle sont connectés respectivement les processeurs d'implantation (ligne 4) et ceux d'interface (ligne 8).

```
1  <composite name="AssemblyFactory">
2    <component name="Implementation">
3      <implementation.java class="ImplementationProcessor"/>
4      <reference multiplicity="0..n" name="implementations"/>
5    </component>
6    <component name="Interface">
7      <implementation.java class="InterfaceProcessor"/>
8      <reference multiplicity="0..n" name="interfaces"/>
9    </component>
10 </composite>
```

Listing 2.5 – Extrait du composite Assembly Factory du module FraSCAti Core

Le module Interface WSDL fournit un fragment d'extension du composite Assembly Factory (voir le listing 2.6). Cette extension est constituée du processeur WSDL (lignes 3 à 6) connecté (ligne 5) à un composant encapsulant le compilateur WSDL vers Java de l'intergiciel Apache CXF. La liaison en ligne 2 exprime que le processeur Interface est connecté au processeur WSDL. Ce fragment de composite ne peut être instancié tout seul car il ne déclare pas le composant Interface. Celui-ci est déclaré dans le fragment principal contenu dans le module FraSCAti Core. Ainsi le module Interface WSDL a une dépendance implicite vers le module FraSCAti Core.

```
1  <composite name="AssemblyFactory">
2    <wire source="Interface/interfaces" target="WSDL"/>
3    <component name="WSDL">
4      <implementation.java class="InterfaceWsdlProcessor"/>
5      <reference name="wsdl-compiler" target="wsdl-compiler/wsdl-compiler"/>
6    </component>
7    <component name="wsdl-compiler">
8      <implementation.java class="WsdlCompilerApacheCXF"/>
9    </component>
10 </composite>
```

Listing 2.6 – Le composite Assembly Factory du module Interface WSDL

fichiers médias, etc.

51. Liaison vers une bibliothèque native en langage C.

Le module `Implementation BPEL` fournit un autre fragment d'extension du composite `Assembly Factory` (voir le listing 2.7). Cette extension est constituée du processeur `BPEL` (lignes 3 à 7) connecté (ligne 5) à un composant encapsulant le moteur EasyBPEL discuté dans la section 2.5.3.1. La liaison en ligne 2 exprime que le processeur `Implementation` est connecté au processeur `BPEL`. La ligne 6 exprime que le composant `BPEL` doit aussi être connecté au composant `wsdl-compiler` car un processus WS-BPEL peut référencer des fichiers WSDL. Ainsi le module `Implementation BPEL` a implicitement besoin des modules `FraSCAti Core` et `Interface WSDL` pour pouvoir être instancié et fonctionné.

```
1  <composite name="AssemblyFactory">
2    <wire source="Implementation/implementations" target="BPEL"/>
3    <component name="BPEL">
4      <implementation.java class="ImplementationBpelProcessor"/>
5      <reference name="bpel-engine" target="bpel-engine"/>
6      <reference name="wsdl-compiler" target="wsdl-compiler"/>
7    </component>
8    <component name="bpel-engine">
9      <implementation.java class="EasyBpelEngine"/>
10   </component>
11 </composite>
```

Listing 2.7 – Le composite `Assembly Factory` du module `Implementation BPEL`

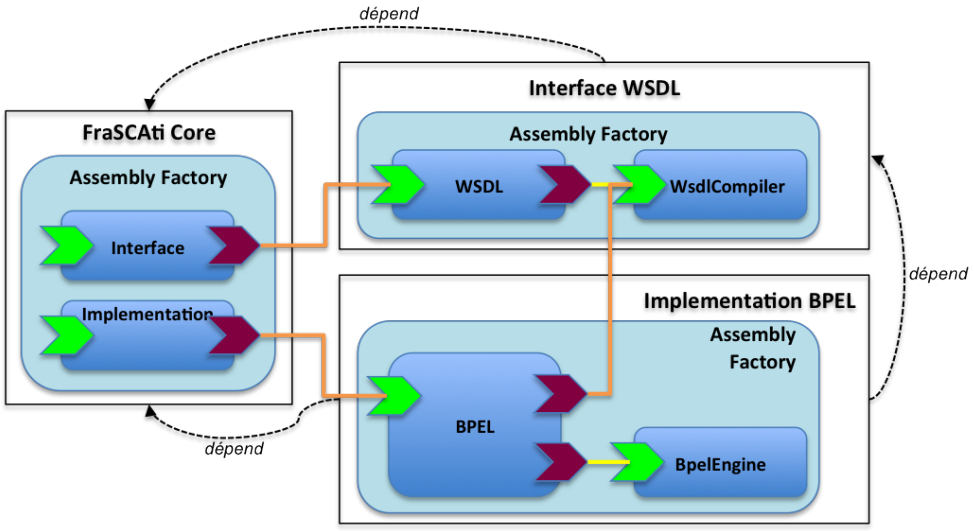

Figure 2.19 – La fragmentation du composite `Assembly Factory`.

La figure 2.19 représente ces trois modules, leur fragment du composite `Assembly Factory` ainsi que leurs dépendances implicites.

2.6.3 La recomposition de l'architecture de la plate-forme FraSCAti

Pour construire " à la carte " une instance de la plate-forme FraSCAti, il suffit de démarrer une machine virtuelle Java avec l'ensemble des modules FraSCAti souhaité, c'est-à-dire lister les archives Java associées à ces modules dans le `classpath` de la machine virtuelle. Ensuite, la plate-forme FraSCAti charge le composite `FraSCAti` et récursivement ces trois sous-composites `Description Parser`, `Assembly Factory` et `Personality Factory`. Lorsque le

composant `SCA Parser` doit charger un composite de nom `C`, il recherche dans le `classpath` l'ensemble des fragments ayant pour nom `C` puis le parseur fusionne tous ces fragments en un seul composite. Ainsi le démarrage de FraSCAti avec les trois modules `FraSCAti Core`, `Interface WSDL` et `Implementation BPEL` produit un composite `Assembly Factory` constitué de l'ensemble des composants contenus dans les trois fragments comme l'illustre la figure 2.20.

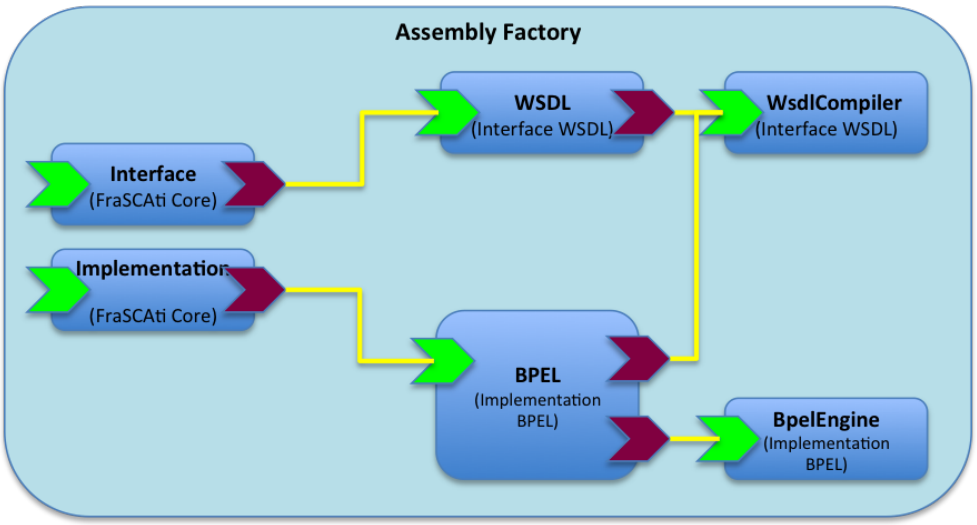

FIGURE 2.20 – La recomposition du composite `Assembly Factory`.

2.6.4 Synthèse et perspectives sur QR2

Grâce à la modularisation de la plate-forme FraSCAti et le mécanisme de fusion des fragments de son architecture, nous avons calculé dans [3] qu'il existe de l'ordre de 10^8 configurations distinctes de la plate-forme FraSCAti version 1.5. A raison d'une minute pour démarrer une configuration, il faudrait de l'ordre de 190 années pour démarrer séquentiellement toutes les configurations de FraSCAti [52].

La fragmentation d'architecture logicielle présentée dans cette section est une technique générique et applicable à n'importe quelle application SCA, c'est-à-dire pas uniquement la fusion des fragments constituant la plate-forme FraSCAti.

Je vois trois perspectives relatives à la modularité de la plate-forme FraSCAti :

1. **Expérimenter la modularisation et fragmentation d'architecture logicielle sur d'autres grandes applications** afin d'évaluer de manière plus générale les bénéfices et les limites de notre approche.
2. **Déduire automatiquement les modules FraSCAti nécessaires à une application métier en introspectant sa description architecturale**. Actuellement, l'architecte doit explicitement lister les modules nécessaires à l'exécution de son application métier, par exemple via des dépendances dans un projet Maven. Pour cela, nous pourrions charger le composite d'une application métier grâce à notre parseur SCA puis

52. J'avoue humblement que je n'ai pas mené cette expérience :-)

visiter l'arbre de syntaxe abstraite produit pour déterminer la liste des constructions SCA utilisées. Grâce à une table associant à chaque construction SCA le module FraSCAti contenant le processeur prenant en charge cette construction, nous pourrions alors en déduire la liste des modules nécessaires à cette application métier.

3. **Charger et décharger dynamiquement des modules FraSCAti durant l'exécution**. Actuellement, il n'est pas possible de charger et décharger dynamiquement à l'exécution des modules de la plate-forme FraSCAti [53]. Charger un nouveau module à l'exécution nécessiterait de reconfigurer l'instance FraSCAti en cours d'exécution afin d'y injecter les nouveaux composants fournis par le dit module. Décharger un module nécessiterait de retirer dynamiquement les composants de ce module de l'instance de FraSCAti en cours d'exécution. Pour cela, il semble intéressant de s'inspirer de la gestion de modules dans les plates-formes OSGi. Toutefois, il faudrait traiter subtilement l'auto-reconfiguration de FraSCAti.

2.7 QR3 : Reconfigurer à l'exécution des applications SCA

Cette section présente notre réponse à la troisième question de recherche (QR3) : comment reconfigurer à l'exécution des applications SCA ?

Dans [171] et [172], Michael P. Papazoglou et al ont identifié que les **architectures reconfigurables dynamiquement à l'exécution** [54] sont le premier défi de recherche pour l'informatique orientée service. Plus spécifiquement dans notre contexte, le standard SCA définit un langage dédié XML pour assembler et configurer des composants pour former des applications orientées services. Ce langage est utilisé durant la conception des applications (voir la section 2.2.2) et est interprété au moment du déploiement de celles-ci (voir la section 2.6). Toutefois, le standard SCA ne spécifie pas les moyens pour reconfigurer les applications durant leur exécution une fois que celles-ci ont été déployées. L'ajout de tels mécanismes de reconfiguration au standard SCA permettrait d'envisager des applications orientées services dynamiquement adaptables voire auto-adaptables [23].

Notre réponse pour reconfigurer à l'exécution des applications SCA repose sur une idée extrêmement simple mais puissante : équiper les composants de capacités réflexives.

2.7.1 Equiper les composants de capacités réflexives

Dans la lignée des modèles de composants OpenCom [26] et Fractal [19], FraSCAti propose un modèle de composants réflexif offrant des capacités d'introspection et de reconfiguration de tout assemblage de composants. Les capacités d'introspection permettent d'examiner un système SCA tandis que celles de reconfiguration permettent de modifier un système SCA. Ces capacités couvrent aussi bien la réflexion structurelle que la réflexion comportementale [217].

Pour cela, chaque composant est équipé d'une interface de programmation réflexive comme l'illustre la figure 2.21. Au dessus de cette interface (voir la section 2.7.2), nous avons construit différents outils réflexifs comme les consoles d'administration FraSCAti Explorer (voir la section 2.7.3), FraSCAti JMX (voir la section 2.7.4) et FraSCAti Web Explorer (voir

53. Toutefois, il est déjà possible de charger et décharger dynamiquement des applications métiers.
54. *"dynamically reconfigurable runtime architectures"* [171, 172]

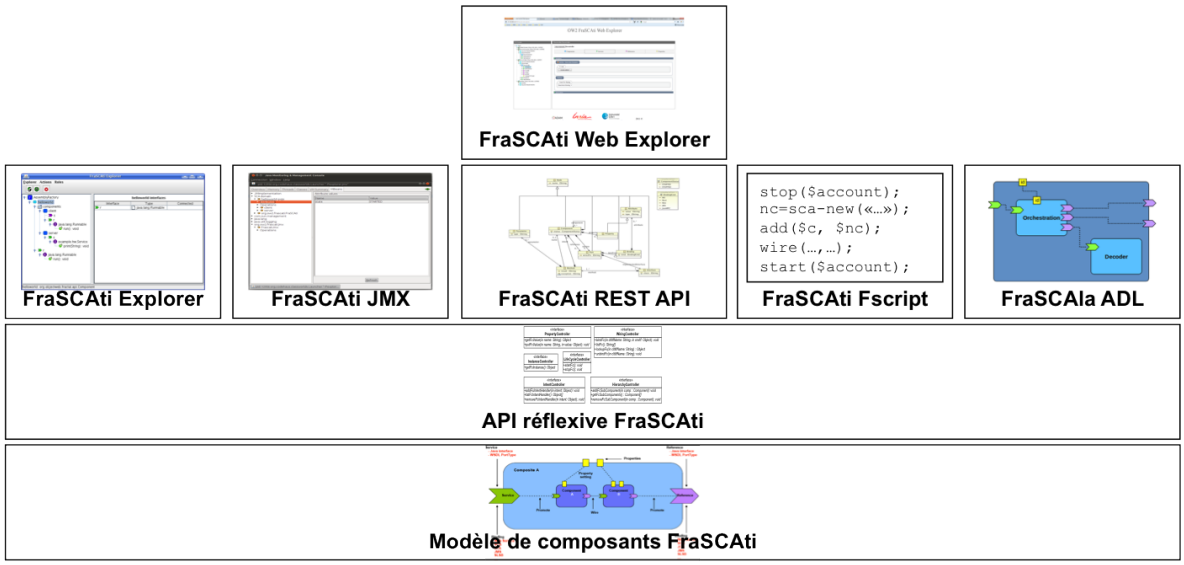

FIGURE 2.21 – La pile réflexive de la plate-forme FRASCATI.

la section 2.7.6), l'interface de programmation FRASCATI REST (voir la section 2.7.5), le langage de script FRASCATI FSCRIPT (section 2.7.7) ainsi que le langage de description d'architecture FRASCALA (voir la section 2.7.8).

2.7.2 L'interface réflexive de FraSCAti

L'interface de programmation réflexive de FRASCATI s'appuie sur celle du modèle de composants FRACTAL. Elle réutilise la notion d'*interface de contrôle* de FRACTAL et plus particulièrement six des sept interfaces de contrôle de FRACTAL : `Component`, `LifeCycleController`, `BindingController`, `ContentController`, `SuperController` et `NameControler`. L'interface vide `AttributeController` de FRACTAL a été remplacée par l'interface `PropertyController` permettant d'introspecter et modifier les propriétés d'un composant SCA. Deux nouvelles interfaces de contrôle ont été introduites. L'interface `IntentController` permet de tisser et détisser dynamiquement des intents sur un composant. L'interface `SCAContentController` permet de modifier dynamiquement l'instance d'objet implantant un composant primitif. L'ensemble de l'interface de programmation réflexive de FRASCATI est disponible à l'adresse `http://frascati.ow2.org/RECONF-API/`.

L'implantation des interfaces de contrôle, c'est-à-dire l'implantation de la réflexivité des composants, est désignée sous le terme de *membrane*. L'originalité de FRASCATI est que **les membranes de composants FraSCAti sont elles-même des assemblages de composants** de contrôle comme l'illustre la figure 2.22. Chaque composant de contrôle implante une interface de contrôle distincte. Par exemple, le composant `HierarchyController` implante l'interface `ContentController`. Chaque composant de contrôle peut nécessiter d'interagir et de se coordonner avec d'autres composants de contrôle. Par exemple, retirer un intent via `IntentController` nécessite que le composant est été stoppé au préalable via le composant de contrôle `LifecycleController` pour éviter de reconfigurer un composant en cours d'utilisation. Pour cela, les composants de contrôle ont des références vers des interfaces de contrôle. Les composants d'aspects (*intents*) font eux-mêmes partis de l'architecture à composants

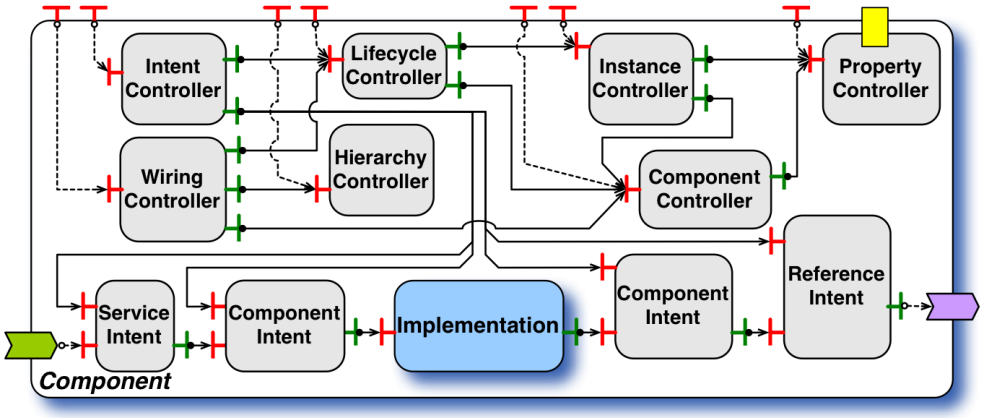

FIGURE 2.22 – L'assemblage des composants de contrôle d'une membrane de composants FRASCATI.

d'une membrane, ils interceptent les appels à destination et en provenance de l'implantation du composant (voir le bas de la figure 2.22).

Cette approche permet de concevoir diverses membranes en fonction des besoins de réflexivité de chaque composant. Par défaut, deux architectures de membranes ont été définies : l'une pour les composants primitifs et l'autre pour les composites. Toutefois, l'utilisateur peut définir ces propres membranes en fonction de ses besoins en retirant et/ou ajoutant de nouveaux composants de contrôle. Par exemple, j'ai défini une extension de la membrane primitive ajoutant une nouvelle interface de contrôle et un nouveau composant de contrôle permettant de manipuler par réflectivité l'état d'un composant. Cette membrane est disponible sous la forme d'un module FRASCATI [55].

La technologie des membranes de FRASCATI s'appelle TINFI et a été entièrement développé par le Prof. Lionel Seinturier. Cette technologie se base sur une approche par compilation des descriptions architecturales des membranes afin de générer la mise en oeuvre de celles-ci. Différents niveaux d'optimisation sont possibles en fonction du niveau de réflexivité versus de performance souhaitée. Par exemple, il est possible de conserver l'architecture à composants de la membrane à l'exécution et ainsi de permettre d'introspecter et reconfigurer les capacités réflexives d'un composant. Dans ce cas, chaque composant de contrôle sera présent à l'exécution et cela aura donc un impact significatif sur l'empreinte mémoire et les performances de la membrane. A l'autre extrême, il est possible de fusionner l'ensemble des composants de contrôle en un unique objet JAVA afin de maximiser les performances et minimiser l'empreinte mémoire de la membrane. Mais ce choix ne permet plus alors d'introspecter et de reconfigurer une membrane.

En terme de performance et d'empreinte mémoire, nous avons montré dans [214] que **l'introduction de la réflexivité, c'est-à-dire l'introspection et la reconfiguration, de composants SCA n'introduit pas de surcoût en termes de temps d'exécution et**

55. `http://websvn.ow2.org/listing.php?repname=frascati&path=/trunk/frascati/modules/frascati-membrane-scaPrimitiveWithState/#`

d'empreinte mémoire par rapport à l'implantation de référence du standard SCA, à savoir Apache Tuscany. La figure 2.23 représente le temps d'invocation et l'empreinte mémoire d'une liste chainée de composants. Chaque composant de la liste ne fait que déléguer les invocations de ses services au composant suivant de la liste. Dans le cas de FraSCAti, nous mesurons ainsi le temps de traversée d'une membrane primitive. Sans entrer dans les détails [56], nous avons observé que les membranes réflexives de FraSCAti sont nettement plus efficaces que la technique de liaison inter-composants mise en oeuvre par Apache Tuscany.

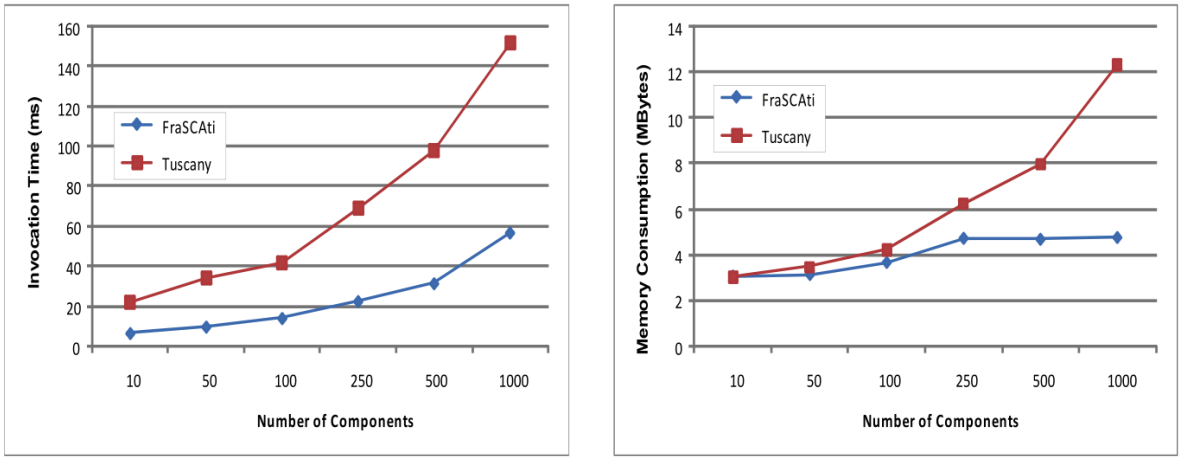

Figure 2.23 – Comparaison de FraSCAti et Apache Tuscany. Source : [214].

2.7.3 L'outil FraSCAti Explorer

Dans le cadre de l'ADT galaxy, Christophe Demarey, ingénieur Inria, et moi avons développé l'outil FraSCAti Explorer. Cet outil propose une interface graphique pour introspecter les composites SCA et reconfigurer les composants SCA comme l'illustre la figure 2.24. On peut ainsi voir cet outil comme un microscope de la réflexivité fournie par FraSCAti. Cet outil s'appuie sur l'interface de programmation réflexive de FraSCAti. Cet outil peut être étendu comme l'illustre la figure 2.12. Cet outil est conditionné sous la forme d'un module FraSCAti et peut ainsi être déployé à la demande de l'utilisateur.

Toutefois, FraSCAti Explorer permet seulement d'administrer localement des applications SCA.

2.7.4 L'outil FraSCAti JMX

L'outil FraSCAti JMX expose les composants SCA sous la forme d'entités administrables à travers l'interface Java Management eXtensions (JMX) [57]. Cet outil réalise le pont entre l'API JMX et l'interface de programmation réflexive de FraSCAti. Ce module FraSCAti a été développé par Nicolas Petitprez, ingénieur sur contrat dans l'équipe ADAM.

Comme l'illustre la figure 2.25, toute application SCA devient administrable par l'intermédiaire de tout outil conforme à JMX, ici la console d'administration JMX livrée en standard

56. Voir [214] pour plus de détails.
57. `http://fr.wikipedia.org/wiki/JMX`

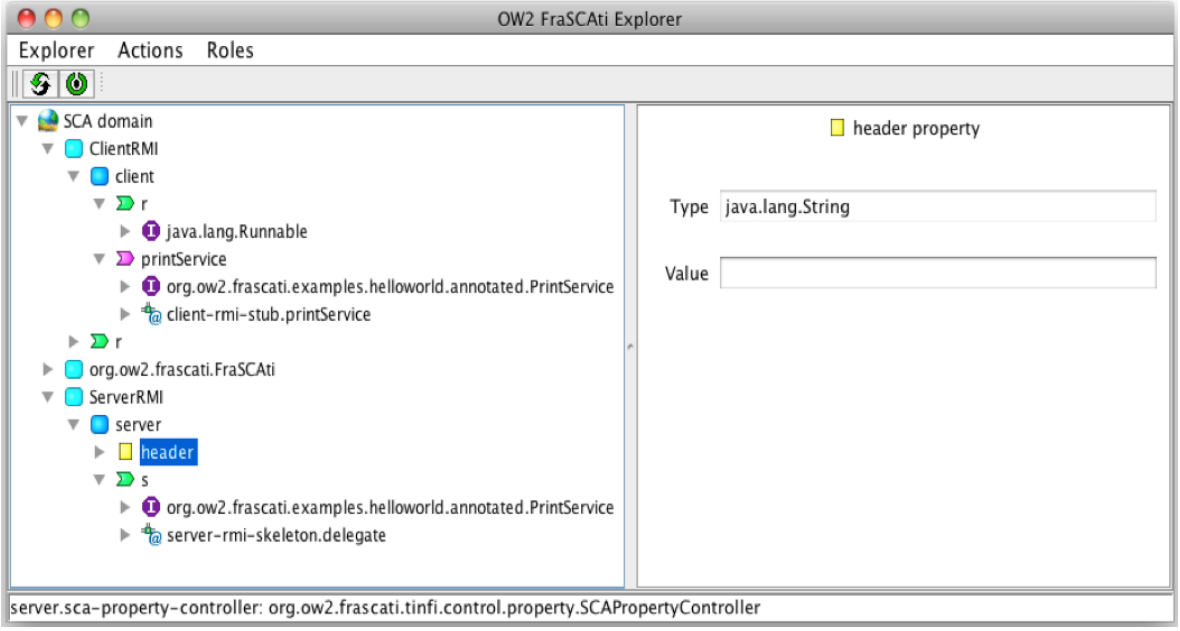

FIGURE 2.24 – FRASCATI EXPLORER : un microscope pour introspecter et reconfigurer des applications SCA.

avec l'environnement d'exécution JAVA. Nous pouvons contacter que l'application FRASCATI en tant que composite SCA devient ainsi administrable via JMX. FRASCATI EXPLORER est réifié par le composant `explorer` tandis que FRASCATI JMX est réifié par le composant `jmx`.

Toutefois, JMX reste un protocole d'administration spécifique au monde JAVA.

2.7.5 L'interface FraSCAti REST

Afin de proposer une administration ouverte et à distance, Christophe Demarey, Gwenaël Cattez et moi avons contribué à une interface REST pour FRASCATI. Cette interface permet de réifier tout composant réflexif FRASCATI sous la forme d'une ressource Web. Comme l'illustre la figure 2.26, il devient ainsi possible de manipuler tout composant FRASCATI depuis tout outil compatible REST tel que par exemple un navigateur Web.

2.7.6 La console FraSCAti Web Explorer

Au dessus de l'interface FRASCATI REST, Gwenaël Cattez et moi avons construit FRASCATI WEB EXPLORER. Cette console permet d'administrer à distance des applications distribuées dans différentes plates-formes FRASCATI.

2.7.7 Le langage FraSCAti FScript

Comme nous venons de le voir, une application s'exécutant sur FRASCATI peut être administrée interactivement via différentes consoles graphiques. Ce type d'outils est tout à fait approprié lorsqu'un administrateur veut réaliser immédiatement des tâches d'adminis-

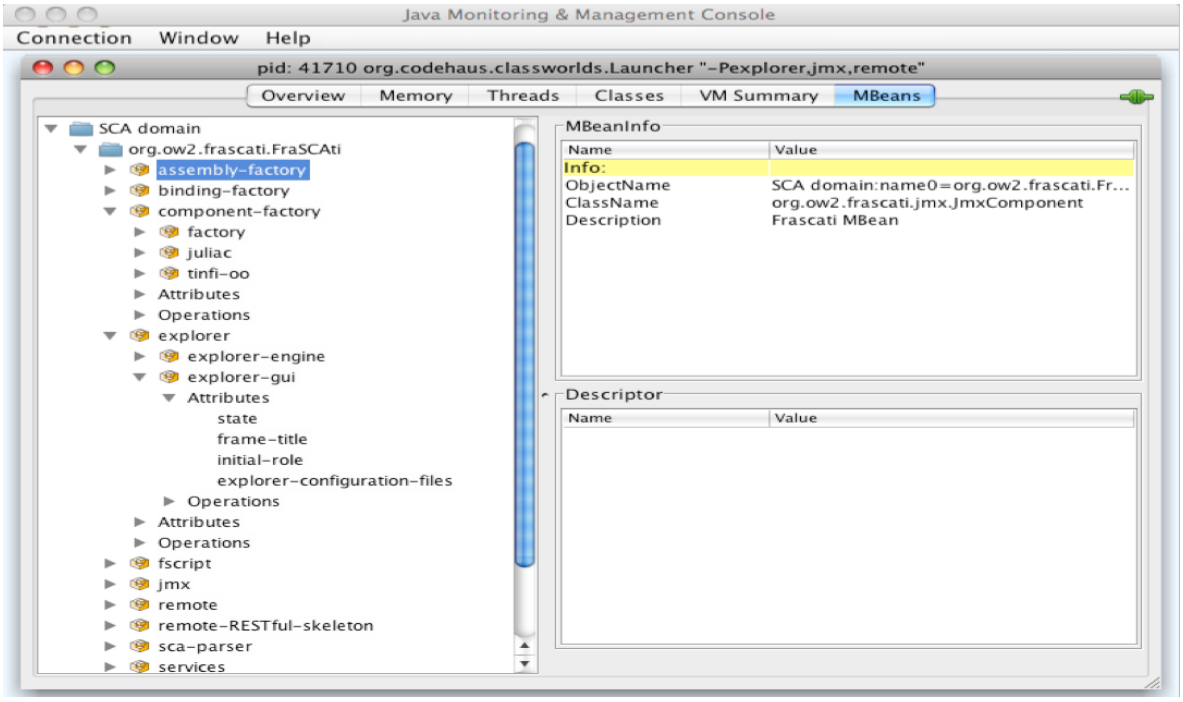

FIGURE 2.25 – FRASCATI JMX.

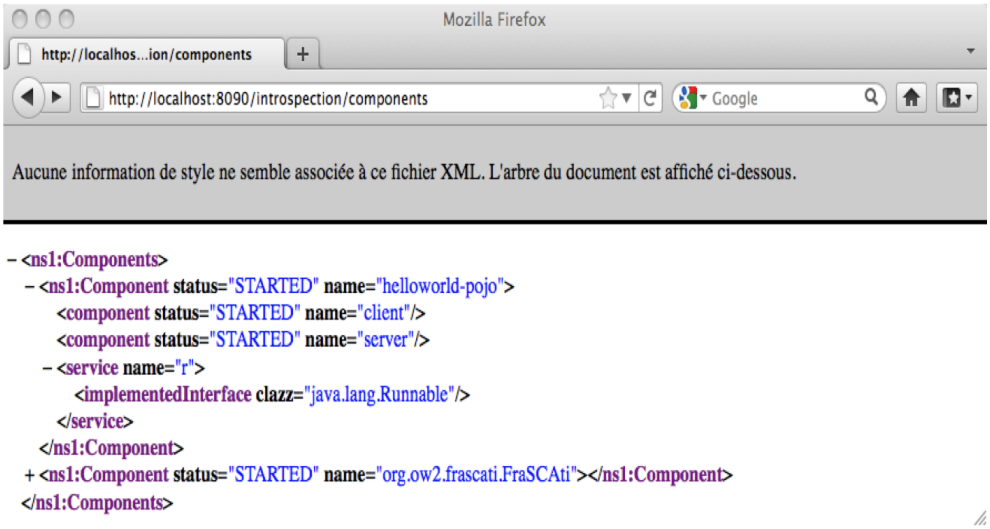

FIGURE 2.26 – L'interface de programmation FRASCATI REST.

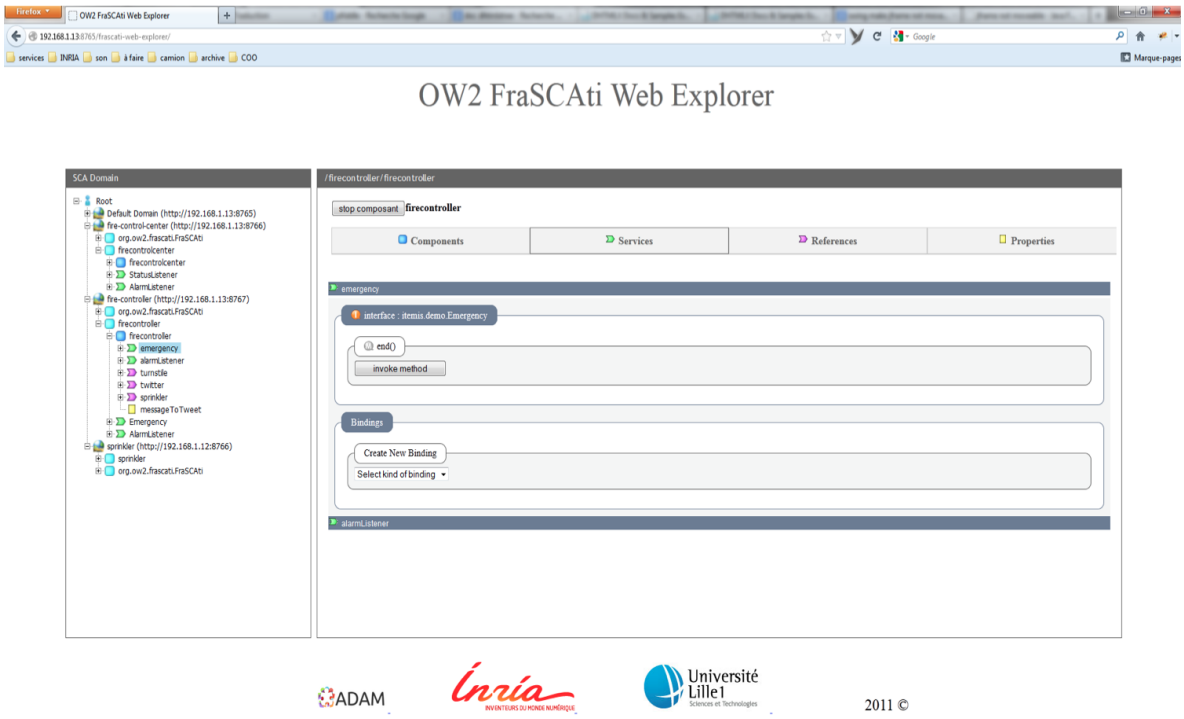

FIGURE 2.27 – FRASCATI WEB EXPLORER.

tration simples et non anticipées. Cependant pour les tâches d'administration complexes et récurrentes, l'administrateur préférerait programmer celles-ci avec un langage de script.

Pour cela, Christophe Demarrey et moi avons réalisé le langage FRASCATI FSCRIPT dans le cadre de l'ADT GALAXY. Ce langage est une extension des langages FPATH et FSCRIPT pour la navigation et la reconfiguration fiable d'architectures FRACTAL [28] qui avaient été proposés dans la thèse de Pierre-Charles David [27].

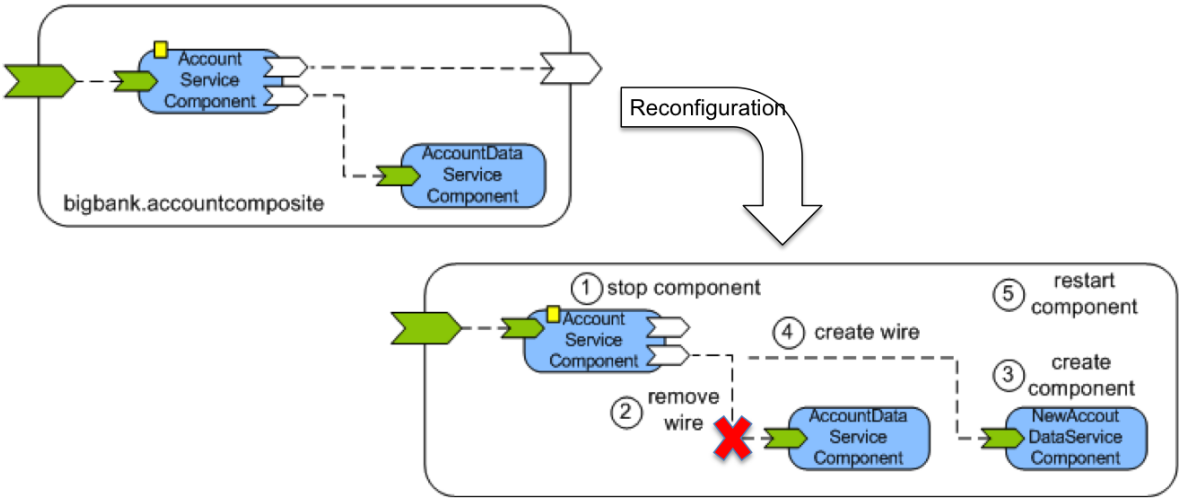

FIGURE 2.28 – Une reconfiguration avec FRASCATI FSCRIPT.

La figure 2.28 illustre un scénario de reconfiguration où le composant `AccountData Service Component` doit être remplacé par un autre composant. Ce scénario nécessite cinq opérations de reconfiguration : 1) stopper le composant `Account Service Component`, 2) rompre la liaison de ce composant vers le composant à remplacer, 3) créer le composant remplaçant, 4) relier le composant stoppé à ce nouveau composant et enfin 5) redémarrer le composant stoppé. Ce scénario s'exprime naturellement en FraSCAti FScript comme l'illustre le listing 2.8. La ligne 1 permet de retrouver le composant `AccountServiceComponent` du composite `MonComposite`. La ligne 2 permet de retrouver la référence à reconfigurer. Les six lignes suivantes réalisent le scénario de reconfiguration décrit précédemment.

```
1 account = $domain/scachild::MonComposite/scachild::AccountServiceComponent;
2 accountRef = $account/scareference::AccountData;
3 stop($account);
4 unwire($accountRef);
5 newAccountData = sca-new("NewAccountDataServiceComponent");
6 add($composite, $newAccountData);
7 wire($accountRef, $newAccountData/scaservice::Data);
8 start($account);
```

Listing 2.8 – Un script de reconfiguration en FraSCAti FScript

Divers usages du langage FraSCAti FScript sont possibles. Une reconfiguration scriptée peut être stockée dans un fichier pour être exécuté à la demande de l'administrateur. La logique métier d'une application peut prévoir des opérations de reconfiguration. Dans ce cas, les reconfigurations seront encapsulées dans des composants applicatifs offrant des services de reconfiguration métier et implantés via des scripts exprimés en FraSCAti FScript. Une autre possibilité est de générer et d'exécuter les scripts de reconfiguration à la volée.

Dans sa thèse [221], Gabriel Tamura propose le système QoS-CARE pour préserver les contrats de qualité de service (QoS) d'applications à composants. Ces applications ainsi que leurs contrats de QoS sont modélisés sous la forme de graphes étiquetés (*e-graph*). Lorsqu'une violation de QoS est observée, QoS-CARE raisonne sur le modèle e-graphe afin de déterminer les reconfigurations nécessaires puis il génère un script de reconfiguration exécuté par l'interpréteur FraSCAti FScript.

2.7.8 Bilan et perspectives sur QR3

Après huit ans de recherche et développement, la réflexivité dans FraSCAti a aujourd'hui atteint un haut niveau de maturité. L'interface de programmation réflexive de FraSCAti est stable et a été utilisée pour bâtir quelques outils réflexifs tels que FraSCAti Explorer, JMX, REST, Web Explorer et FScript. La réflexivité de FraSCAti a été mise en oeuvre dans plusieurs travaux de recherche sur les systèmes (auto-)adaptables tels que CALICO [240], CAPucine [180], SPACES [197], QoS-CARE [221], Ceviche [66], CORONA [142] ainsi que [7], pour n'en citer que quelque uns.

Depuis octobre 2014, j'ai démarré une collaboration avec la Prof. Olga Kouchnarenko et le doctorant Jean-François Weber de l'équipe VESONTIO du Laboratoire d'Informatique de Franche-Comté. L'objectif est de porter leur logique temporelle pour l'expression de politiques de reconfiguration de systèmes à composants [75] au dessus de la plate-forme FraSCAti. Un premier portage a déjà été prototypé avec succès. Une prochaine étape sera d'intégrer leur langage dédié d'expression de politiques temporelles de reconfiguration au sein de notre langage FraSCAla.

FraSCAla est un langage de description d'architectures auto-adaptables sur lequel Romain Rouvoy et moi réfléchissons depuis plusieurs années. Ce langage permet simultanément de décrire l'architecture à composants d'un système logiciel, l'implantation des composants métier ainsi que les règles d'auto-adaptation. FraSCAla devrait à terme permettre d'exprimer l'auto-adaptation selon différents paradigmes simultanément comme par exemple via des règles ECA [58] intégrant une logique temporelle, des aspects et/ou des patrons d'architecture. Un prototype de ce langage a été réalisé en tant que langage dédié (DSL) embarqué dans le langage généraliste Scala. Par compilation, un programme FraSCAla est transformé en un ensemble de composants exécutables au dessus de la plate-forme FraSCAti. Ainsi FraSCAla s'inscrit dans la mouvance des langages de programmation d'architectures initiée par ArchJava [5]. Toutefois, le langage ArchJava ne permet pas d'exprimer l'auto-adaptation d'architectures à composants. Le papier [207] présente une première ébauche du langage FraSCAla.

2.8 QR4 : Modéliser la variabilité de la plate-forme FraSCAti

Cette section présente notre réponse à la quatrième question de recherche (QR4) : comment modéliser la variabilité d'une plate-forme SCA ?

Une plate-forme SCA doit nativement être extensible pour supporter diverses technologies d'implantation des composants (`<implementation.*>`), de description des interfaces (`<interface.*>`), de liaison d'interopérabilité (`<binding.*>`) et de propriétés extra-fonctionnelles (*intents*). Nous avons vu précédemment que la plate-forme FraSCAti supporte cette extensibilité grâce à une architecture modulaire à base de composants. La plate-forme FraSCAti offre un nombre significatif de modules composables à la demande. L'utilisateur peut alors construire différentes instances de la plate-forme FraSCAti en fonction des besoins de ses applicatifs mais aussi des contraintes de l'environnement d'exécution. Toutefois, ces modules peuvent être incompatibles entre eux. Par exemple, FraSCAti propose différents modules pour implanter des composants via des bundles OSGi ou via des archives EJB. Cependant, une configuration / instance de FraSCAti ne peut contenir au plus qu'un seul module OSGi et un seul module EJB. D'un autre coté, certains modules ont des dépendances, par exemple le module WS-BPEL utilise le module WSDL. Cette variabilité est encodée dans l'architecture modulaire de la plate-forme FraSCAti comme l'illustre la figure 2.29. Il conviendrait toutefois de pouvoir modéliser cette variabilité afin de pouvoir raisonner sur celle-ci.

Notre réponse pour modéliser la variabilité de la plate-forme FraSCAti repose sur deux idées extrêmement simples mais puissantes : considérer FraSCAti comme une ligne de produits logiciels et modéliser la variabilité de FraSCAti via un modèle de fonctionnalités.

2.8.1 La modélisation des fonctionnalités de la plate-forme FraSCAti

Une ligne de produits logiciels ou *Software Product Line* peut être définie comme "*a set of software-intensive systems that share a common, managed set of features and that are developed from a common set of core assets in a prescribed way*" [25]. Selon cette définition,

58. Evénement - Condition - Actions

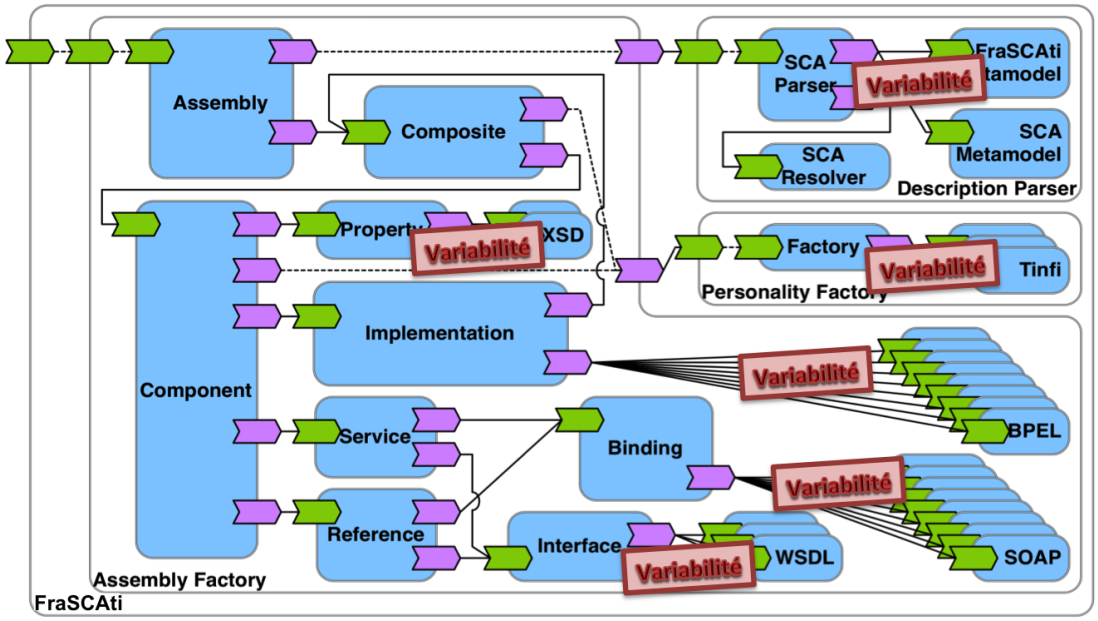

FIGURE 2.29 – Les points de variabilité de l'architecture de FRASCATI.

FRASCATI peut tout à fait être considéré comme une ligne de produits logiciels. Un modèle de fonctionnalités ou *Feature Model* est généralement utilisé pour définir de manière compacte toutes les fonctionnalités d'une ligne de produits logiciels aussi bien que leurs combinaisons valides [212]. Un modèle de fonctionnalités peut être représenté graphiquement par un diagramme de fonctionnalités. La figure 2.30 présente un extrait du diagramme de fonctionnalités de FRASCATI. Un diagramme de fonctionnalités a une structure arborescente. La racine représente la ligne de produits, ici la fonctionnalité `FraSCAti`. Chaque fonctionnalité peut avoir des fonctionnalités filles obligatoires (*mandatory*) telles que `SCAParser`, `Assembly Factory`, `Component Factory`[59], `Metamodel`, `Binding`, `Java Compiler` et des fonctionnalités optionnelles (*optional*) telles que `MMFrascati` et `MMTuscany`. Des fonctionnalités peuvent être regroupées pour exprimer qu'elles sont compatibles telles que `http` et `rest` ou qu'elles sont exclusives telles que `JDK6` et `JDT`. Enfin il est possible d'exprimer des contraintes de dépendances entre fonctionnalités telles que `rest` nécessite `MMFrascati` et `http` nécessite `MMTuscany`.

Comme discuté dans [212], un diagramme de fonctionnalités a une sémantique précise en logique propositionnelle du premier ordre[60]. La formule booléenne suivante encode précisément le diagramme de la figure 2.30.

59. Synonyme du terme *Personality Factory* utilisé précédemment.
60. `http://en.wikipedia.org/wiki/Feature_model#Semantics`

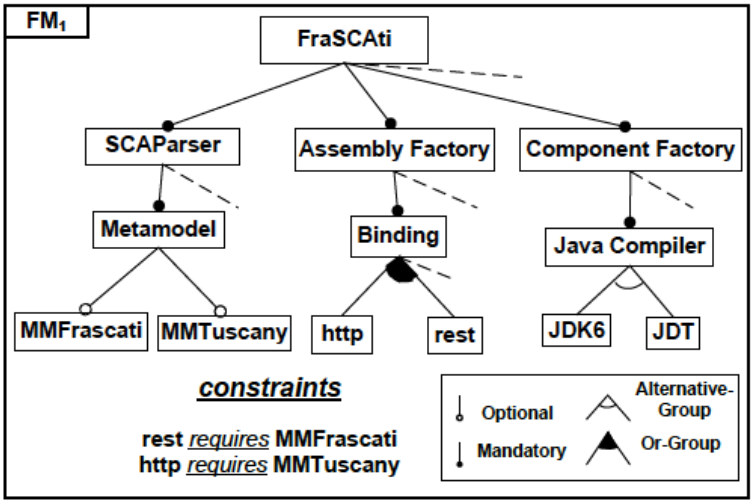

FIGURE 2.30 – Un extrait du diagramme de fonctionnalités de FRASCATI. Source : [2].

$$
\begin{aligned}
FM = {} & FraSCAti \\
& \wedge FraSCAti \iff AssemblyFactory \\
& \wedge FraSCAti \iff ComponentFactory \\
& \wedge FraSCAti \iff SCAParser \\
& \wedge SCAParser \iff Metamodel \\
& \wedge AssemblyFactory \iff Binding \\
& \wedge ComponentFactory \iff JavaCompiler \\
& \wedge JavaCompiler \Rightarrow JDK6 \vee JDT \\
& \wedge \neg JDK6 \vee \neg JDT \\
& \wedge MMFrascati \Rightarrow Metamodel \\
& \wedge MMTuscany \Rightarrow Metamodel \\
& \wedge http \Rightarrow Binding \\
& \wedge rest \Rightarrow Binding \\
& \wedge Binding \Rightarrow rest \vee http \\
& \wedge rest \Rightarrow MMFrascati \\
& \wedge http \Rightarrow MMTuscany
\end{aligned}
$$

Un diagramme de fonctionnalités peut soit être simplement dessiné comme c'est le cas de la figure 2.30, soit être encodé via un outil dédié à l'édition et la manipulation de diagrammes de fonctionnalités. Le diagramme complet des fonctionnalités de FRASCATI version 1.4 est présenté dans la figure 2.31 et fut manuellement encodé dans l'outil S2T2 développé au LERO [61]. L'outil S2T2 permet de visualiser des diagrammes de fonctionnalités mais aussi de créer des configurations valides (les fonctionnalités cochées et en couleur vert dans la figure 2.31).

Toutefois, définir manuellement le modèle de fonctionnalités de la plate-forme FRASCATI peut introduire des erreurs de modélisation des relations entre fonctionnalités (c'est-à-dire

61. `http://download.lero.ie/spl/s2t2/index.html`

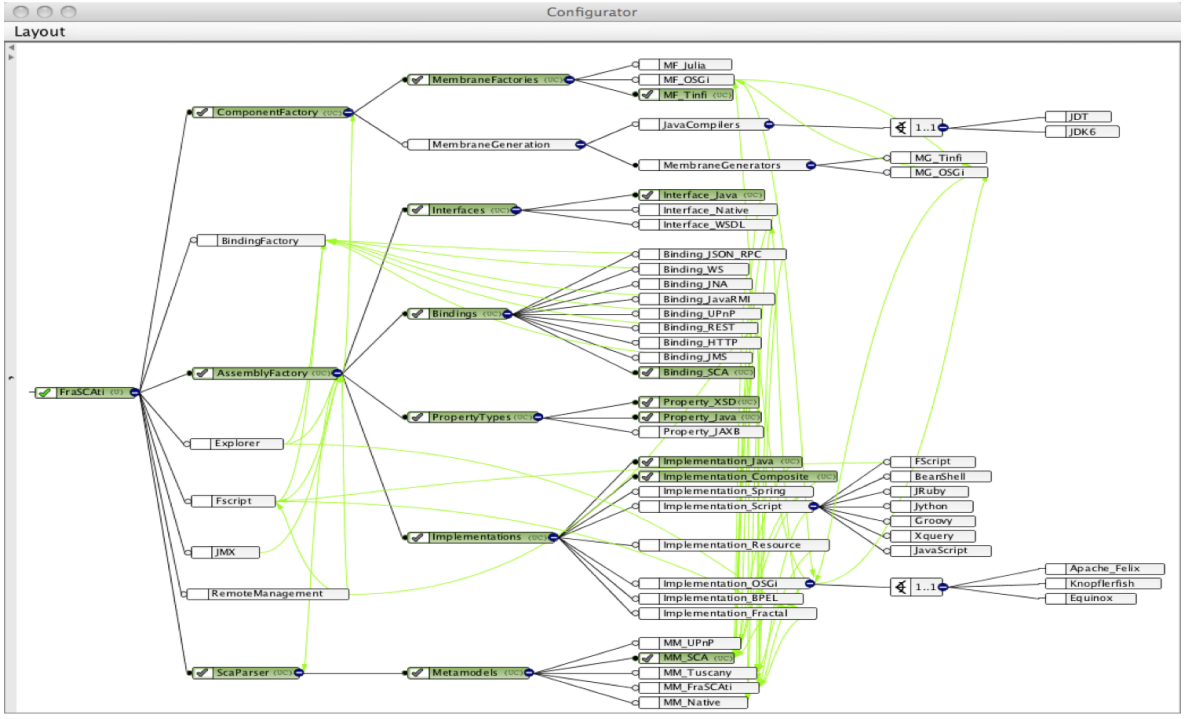

FIGURE 2.31 – Le diagramme de fonctionnalités de FRASCATI 1.4.

obligatoire versus optionnelle, groupe inclusif versus exclusif, contrainte de dépendances) ou des oublis de fonctionnalités “ cachées ” au sein de l'architecture de FRASCATI. De plus comme la plate-forme FRASCATI évolue en permanence, e.g., de nouveaux modules et/ou composants sont ajoutés, des réorganisations de l'architecture à composants peuvent s'avérer nécessaires, il devient difficile de maintenir le modèle de fonctionnalités en cohérence avec l'architecture modulaire de la plate-forme FRASCATI.

2.8.2 L'extraction automatique du modèle de fonctionnalités de la plate-forme FraSCAti

Pour pallier à ce problème de maintenance, nous avons développé une procédure d'extraction automatique du modèle de fonctionnalités de la plate-forme FRASCATI en collaboration avec l'université de Nice Sophia-Antipolis. Cette procédure a été présentée à la conférence européenne sur les architectures logicielles (ECSA) [2] puis publiée dans le journal *Software and Systems Modeling* (SoSyM) [3].

Comme l'illustre la figure 2.32, cette procédure d'extraction prend en entrée l'ensemble des artefacts logiciels de FRASCATI, c'est-à-dire tous les modules FRASCATI.

Premièrement, un modèle de fonctionnalités architecturales ($fm_{Arch150}$) est construit à partir de l'analyse de la description architecturale de FRASCATI, c'est-à-dire à partir de la fusion de tous les fragments de composites SCA. Quelques règles de production de ce modèle sont illustrées dans la figure 2.33. L'analyse démarre sur le composite `FraSCAti` et crée la fonctionnalité racine associée. Ensuite, chaque composant du composite est analysé. Si l'implantation du composant est un composite alors une fonctionnalité fille obligatoire est créée, comme par exemple pour le composant `sca-parser`, puis ce composite est analysé récursive-

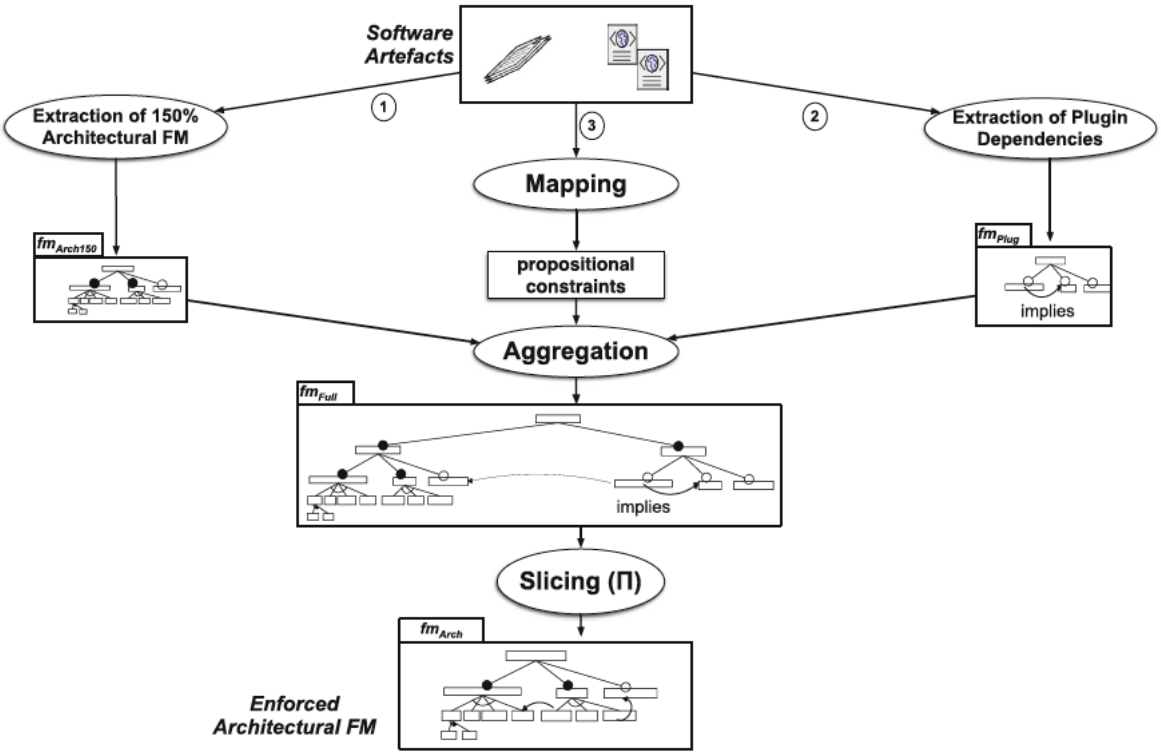

FIGURE 2.32 – Le processus d'extraction du modèle de fonctionnalités de FRASCATI. Source : [3].

ment. Si le composant a une référence multiple (`multiplicity=0..n`) alors une fonctionnalité fille optionnelle est créée, comme c'est le cas pour la référence `metamodels`. Ensuite, chaque liaison (`wire`) du composite est analysée. Si la liaison a pour source une référence multiple alors une fonctionnalité fille inclusive est créée pour le composant destination, comme c'est le cas pour le composant `sca-metamodel`.

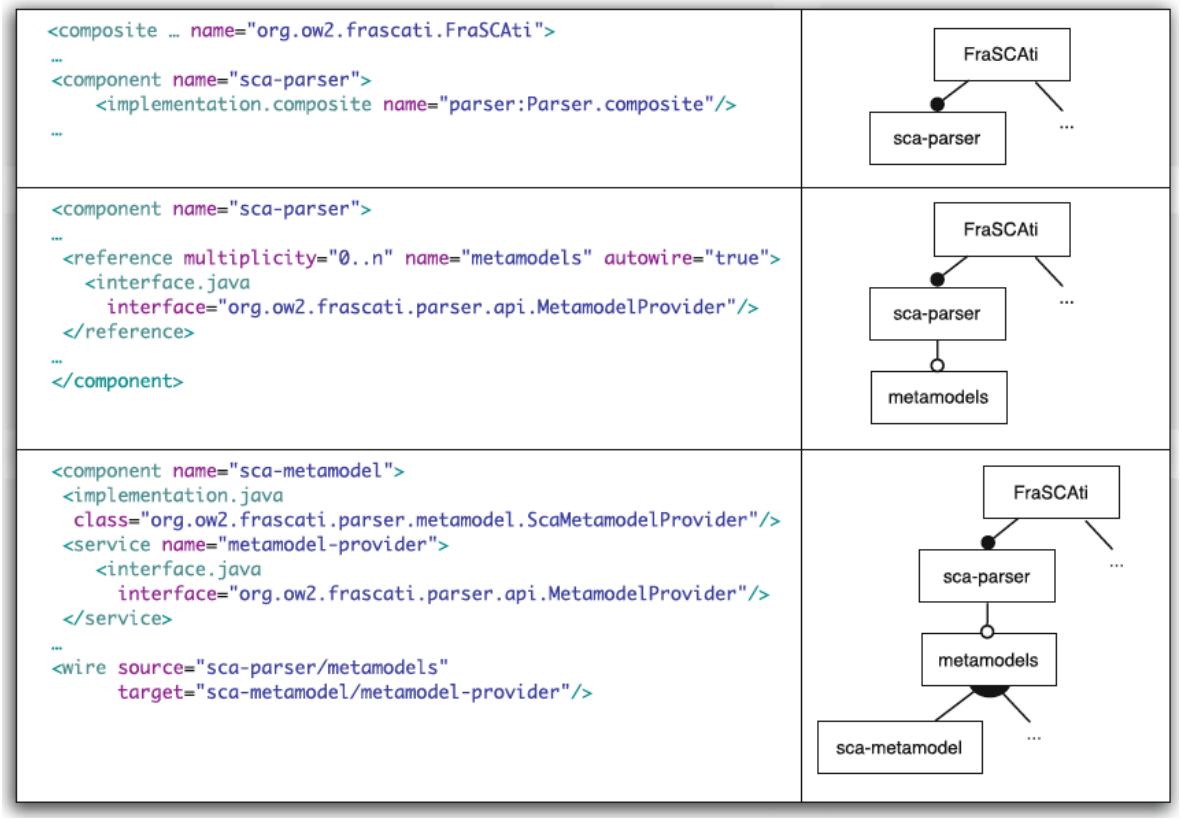

FIGURE 2.33 – L'extraction des fonctionnalités depuis l'architecture de FRASCATI. Source : [3].

Deuxièmement, un modèle de fonctionnalités (fm_{Plug}) est construit pour représenter les modules et leurs dépendances MAVEN. Chaque module est représenté par une fonctionnalité et les dépendances MAVEN sont représentées par des contraintes *requires*. Par exemple, la fonctionnalité `BPEL` dépend des fonctionnalités `FraSCAti Core` et `WSDL`.

Troisièmement, un ensemble de contraintes d'implication (*implies*) est généré. Ces contraintes permettent d'exprimer quel est le module contenant une fonctionnalité architecturale et quels sont les fonctionnalités contenues dans un module. Par exemple, la fonctionnalité module `BPEL` implique la fonctionnalité processeur `BPEL`. Ces contraintes permettent ainsi de définir les relations d'implication entre le modèle de fonctionnalités architecturales et le modèle de fonctionnalités des plugins.

Ensuite, les modèles $fm_{Arch150}$ et fm_{Plug} ainsi que les contraintes d'implication sont agrégés pour former le modèle de fonctionnalités complet (fm_{Full}). Finalement, le modèle fm_{Full} est taillé (*slice*) pour ne conserver que les fonctionnalités architecturales.

L'article [3] détaille plus précisément les différents algorithmes mis en place ainsi que notre

approche pour valider que ces algorithmes produisent les résultats escomptés.

Grâce à cette procédure, nous avons pu générer automatiquement le modèle de fonctionnalités de FraSCAti version 1.4. Ce modèle a été publié sur le site *Software Product Lines Online Tools*[62] (SPLOT) [111]. SPLOT est le site de référence dans le domaine des modèles de fonctionnalités. Ce site fournit des outils en ligne pour éditer et analyser des modèles de fonctionnalités, puis configurer des produits. La figure 2.34 montre une capture de l'éditeur en ligne de modèle de fonctionnalités appliqué sur FraSCAti.

Au 1 juin 2015, le site SPLOT fournissait un référentiel de 657 modèles de fonctionnalités et notre modèle de fonctionnalités OW2 FraSCAti 1.4 occupait la 34ième place en termes de nombre de fonctionnalités incluses dans le modèle.

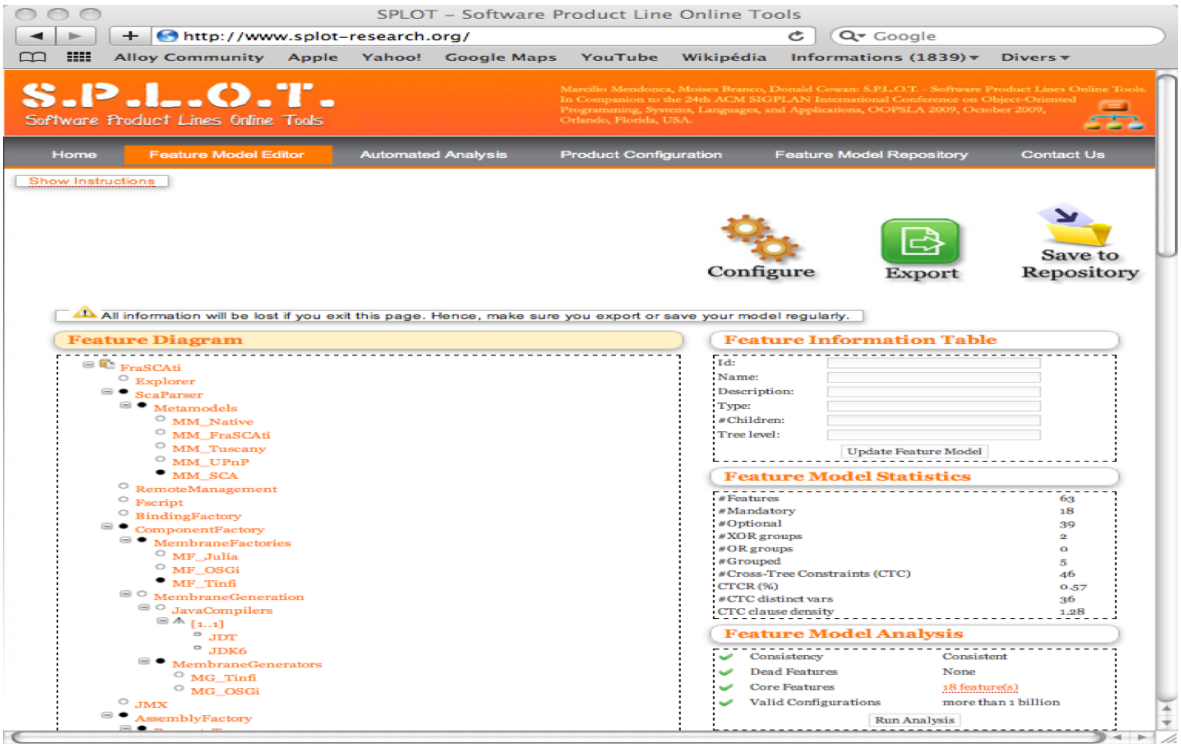

FIGURE 2.34 – Le modèle de fonctionnalités de FraSCAti publié sur le site S.P.L.O.T..

Le modèle de fonctionnalités de FraSCAti totalise 63 fonctionnalités dont 18 obligatoires et 45 optionnels. 33 fonctionnalités couvrant diverses types d'implantation, d'interface, de liaison d'interopérabilité sont directement destinées aux développeurs d'applications SCA. Cinq fonctionnalités représentent les outils réflexifs offerts par FraSCAti 1.4 et sont à destination des utilisateurs de la plate-forme FraSCAti. Enfin 25 fonctionnalités couvrent des aspects internes de la plate-forme, à savoir les méta-modèles supportés, les générateurs de membranes, les compilateurs Java et fabriques de membranes. Au total, il existe plus d'un milliard de configurations valides et distinctes de la plate-forme FraSCAti 1.4. Depuis la version 1.4, de nombreuses nouvelles fonctionnalités ont été ajoutées ce qui devrait avoir exponentiellement augmenté le nombre de configurations valides.

62. `http://www.splot-research.org/`

2.8.3 L'évolution du modèle de fonctionnalités de la plate-forme FraSCAti

Dans notre article SoSym [3], nous proposons d'aller plus loin en raisonnant sur l'évolution du modèle de fonctionnalités d'une ligne de produits telle que FraSCAti. Nous pouvons appliquer la procédure d'extraction sur chaque version de la ligne de produits FraSCAti comme l'illustre la figure 2.35. Ensuite nous calculons la différence entre les modèles de fonctionnalités $n-1$ et n afin de déterminer la liste des nouvelles fonctionnalités et de celles supprimées ou modifiées. Notre algorithme de calcul de différence entre modèles a été présenté à la conférence internationale CAiSE [4].

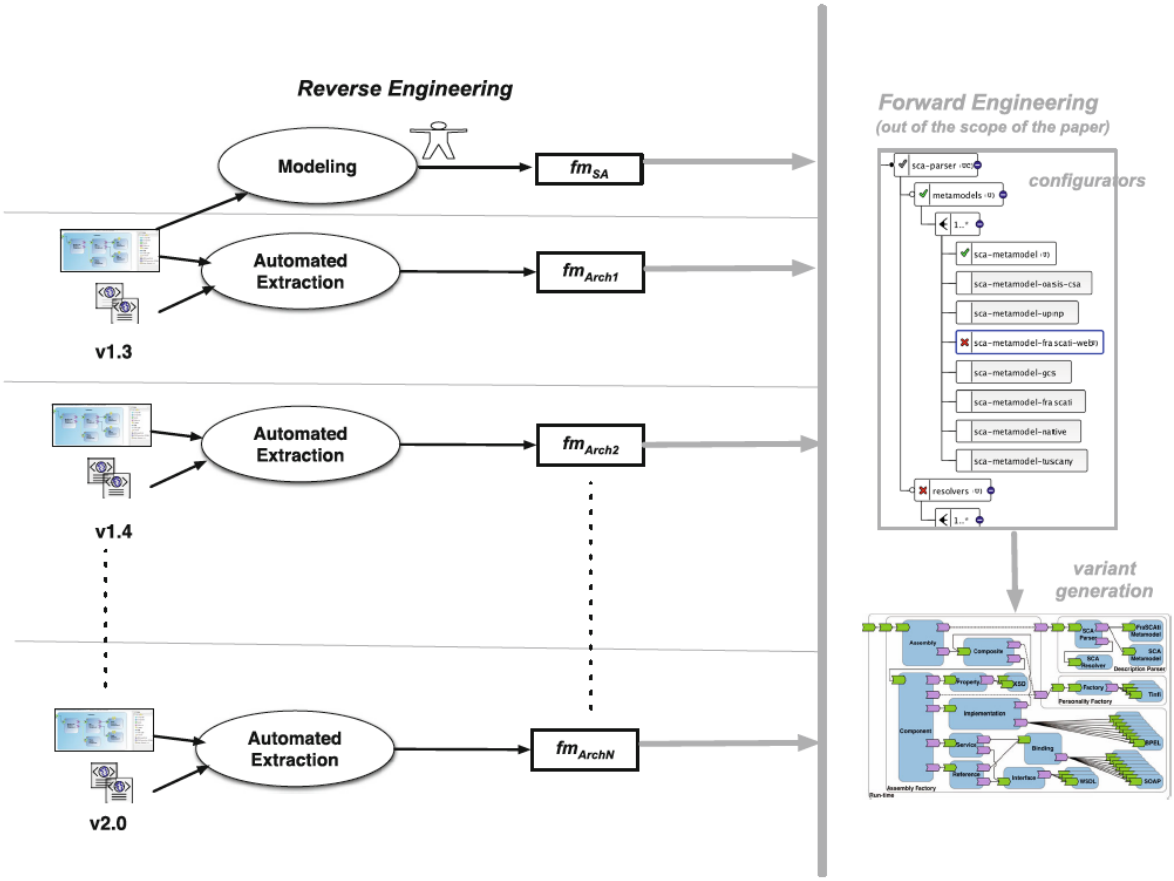

FIGURE 2.35 – L'évolution du modèle de fonctionnalités de FraSCAti. Source : [3].

Pour aller encore plus loin, nous pourrions intégrer ce processus de suivi de l'évolution du modèle de fonctionnalités d'une ligne de produits logiciels au processus d'intégration continue. A chaque changement dans l'architecture modulaire de la ligne de produits logiciels, e.g., modification des descripteurs d'assemblage de composants et de modules, il serait possible d'évaluer l'impact sur la variabilité de la ligne de produits logiciels. Ainsi la variabilité et son évolution pourraient devenir de nouveaux indicateurs pour évaluer en continu la qualité de grands logiciels complexes.

2.9 Synthèse sur FraSCAti

Afin d'adresser l'adaptabilité de systèmes distribués et plus particulièrement dans le contexte de l'informatique orientée service, nous avons proposé la plate-forme intergicielle FraSCAti. Voici une synthèse des principaux traits de conception à retenir :

Un intergiciel d'intergiciels (*cf.*, QR1)

Afin de prendre en compte l'hétérogénéité et la diversité des paradigmes d'interactions distribuées, des intergiciels, de leurs interfaces de programmation, de leurs protocoles de transport, je propose une nouvelle organisation de la couche intergicielle désignée sous le vocable de *intergiciel d'intergiciels* ou *middleware of middleware*. Un intergiciel d'intergiciels est un intergiciel intégrant d'autres intergiciels hétérogènes et proposant une interface unique aux applications, comme l'illustre la figure 2.36.

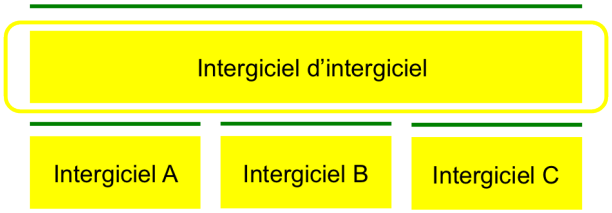

Figure 2.36 – Un intergiciel d'intergiciels.

Selon cette définition, FraSCAti est un intergiciel d'intergiciels car il intègre divers intergiciels hétérogènes (services Web et REST via Apache CXF, Java RMI, JMS, JGroups, EJB, OSGi, WS-BPEL, etc.) et il propose aux applications une interface unique inspirée du standard SCA.

Des composants à tous les étages (*cf.*, transverse QR1/QR2/QR3/QR4)

Notre travail s'inscrit dans la démarche visant à promouvoir les composants comme entité unificatrice pour le développement de systèmes distribuées allant des couches applicatives aux couches intergicielles et systèmes.

Dans FraSCAti, nous proposons une unification des Services, Composants et Aspects (SCA) en promouvant des composants à tous les étages comme l'illustre la figure 2.37 :

- les *composants applicatifs*,
- les *composants d'aspect* pour les propriétés extra-fonctionnelles,
- les *composants de liaison* d'interopérabilité,
- les *composants de la plate-forme* FraSCAti,
- les *composant de l'intergiciel* EasyBPEL,
- les *composants de contrôle* des membranes réflexives.

Un intergiciel réflexif (*cf.*, QR3)

Notre travail s'inscrit dans la mouvance des intergiciels réflexifs tels que OpenCom et Fractal. Tout composant FraSCAti est équipé de capacités réflexives d'introspection, d'intercession et de reconfiguration. Ainsi les applications ainsi que la couche intergicielle sont dynamiquement adaptables voire auto-adaptables. La réflexivité est

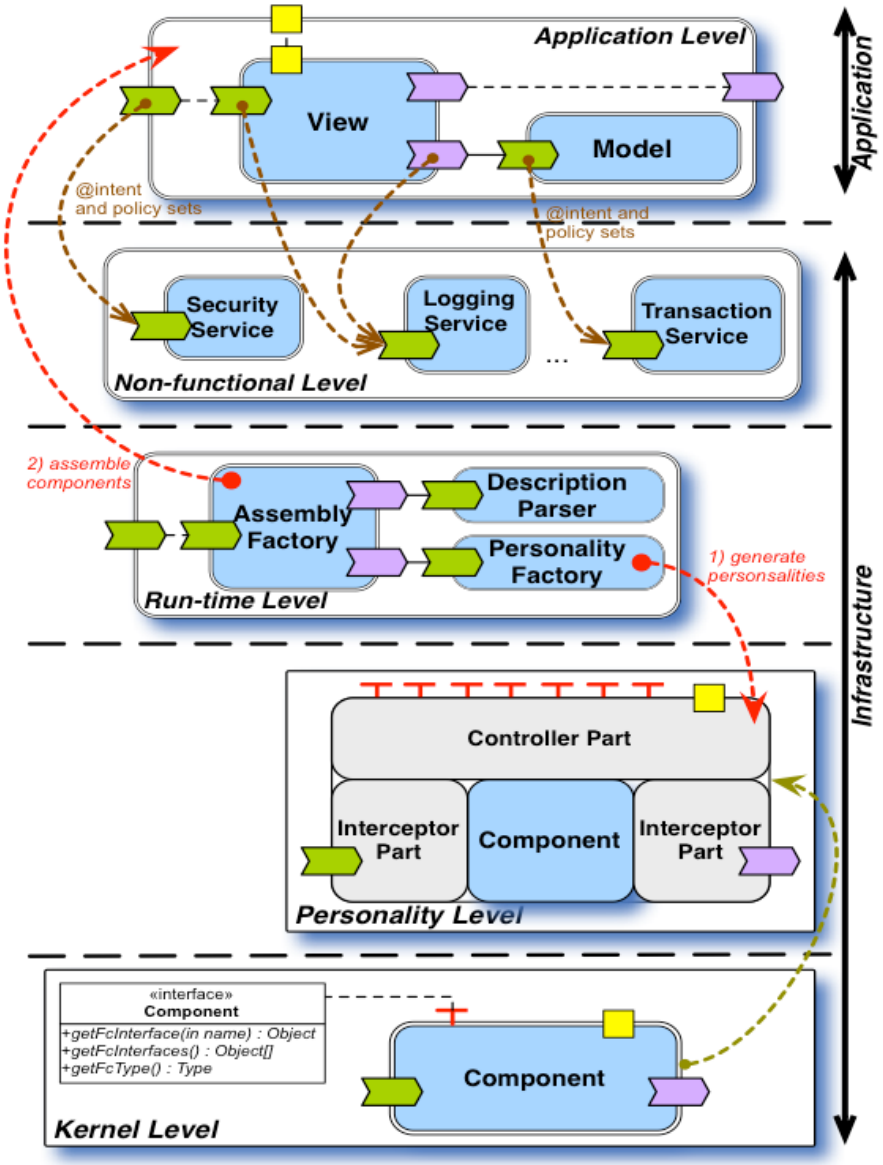

FIGURE 2.37 – Des composants à tous les étages.

prise en charge par la notion de *membrane* constituée d'assemblage de composants de contrôle permettant de mettre en oeuvre différentes formes de réflexivité structurelle ainsi que comportementale.

Une ligne de produits intergiciels (*cf.*, QR2/QR4)

Comme tout grand logiciel complexe, un intergiciel d'intergiciels propose un très grand nombre de fonctionnalités. Toutefois, il convient de pouvoir sélectionner le sous-ensemble de fonctionnalités strictement nécessaires aux besoins applicatifs et/ou au contexte d'exécution. Pour cela, nous proposons de construire un intergiciel d'intergiciels comme une ligne de produits intergiciels ou *Middleware Product Line* hautement modulaire, configurable et extensible.

Une telle ligne de produits doit fournir un modèle de fonctionnalités en permanence causalement connecté à l'architecture modulaire de la ligne de produits. L'implantation de la ligne doit être décomposée en modules contenant chacun un fragment de l'architecture globale et pour les modules optionnels au mieux l'implantation d'une seule fonctionnalité optionnelle.

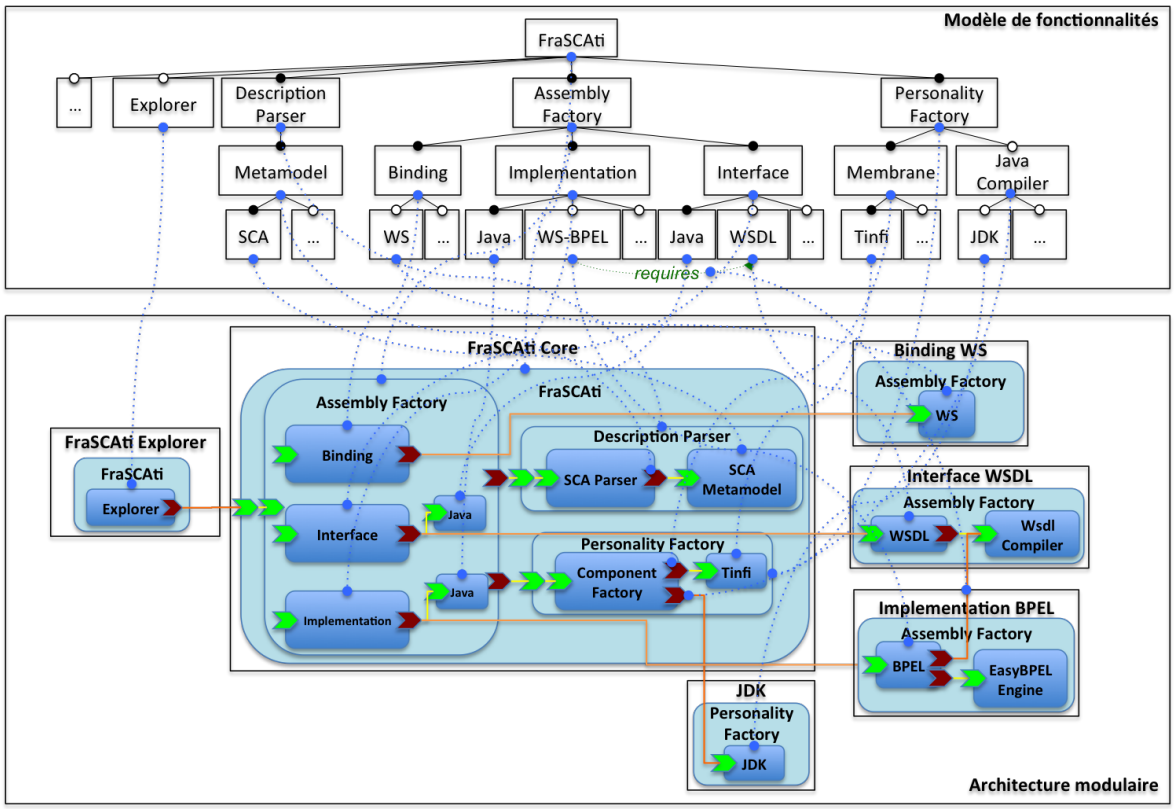

FIGURE 2.38 – Le lien causal entre le modèle de fonctionnalités et l'architecture modulaire de FRASCATI.

La figure 2.38 présente un sous-ensemble du diagramme de fonctionnalités et de l'architecture modulaire de la plate-forme FRASCATI, c'est-à-dire 19 des 63 fonctionnalités offertes et 6 des 46 modules constituant la plate-forme FRASCATI version 1.4. Les

connecteurs pointillés en bleu représentent les liens de causalités reliant une fonctionnalité au composant l'implantant. Les composants du module `FraSCAti Core` sont des fonctionnalités obligatoires, à savoir `Description Parser`, `Metamodel`, `SCA Metamodel`, `Assembly Factory`, `Binding`, `Interface` et `Java`, `Implementation` et `Java`, `Membrane` et `Tinfi`. Les quatre autres modules contiennent chacun l'implantation d'une fonctionnalité optionnelle, à savoir l'outil `Explorer`, le `binding WS`, l'`interface WSDL`, l'`implementation WS-BPEL` et le compilateur du `JDK`.

2.10 L'orchestration décentralisée et dynamique de processus métiers à base de services

Cette section résume le contexte, la problématique et les contributions de la thèse de Virginie Legrand Contes [87] que j'ai co-encadrée avec la Prof. Françoise Baude de l'équipe-projet Inria OASIS [63] à l'Université de Nice - Sophia Antipolis.

2.10.1 Le contexte

La thèse de Virginie Legrand Contes s'est déroulée dans le contexte du projet européen SOA4All [64] portant sur les *architectures orientées services pour tous*. Le projet se basait sur l'hypothèse qu'à terme l'Internet allait devenir un vaste *réseau de milliards de services* qui nécessiterait une plate-forme mondiale pour la livraison, la recherche, la consommation et le suivi de ces services.

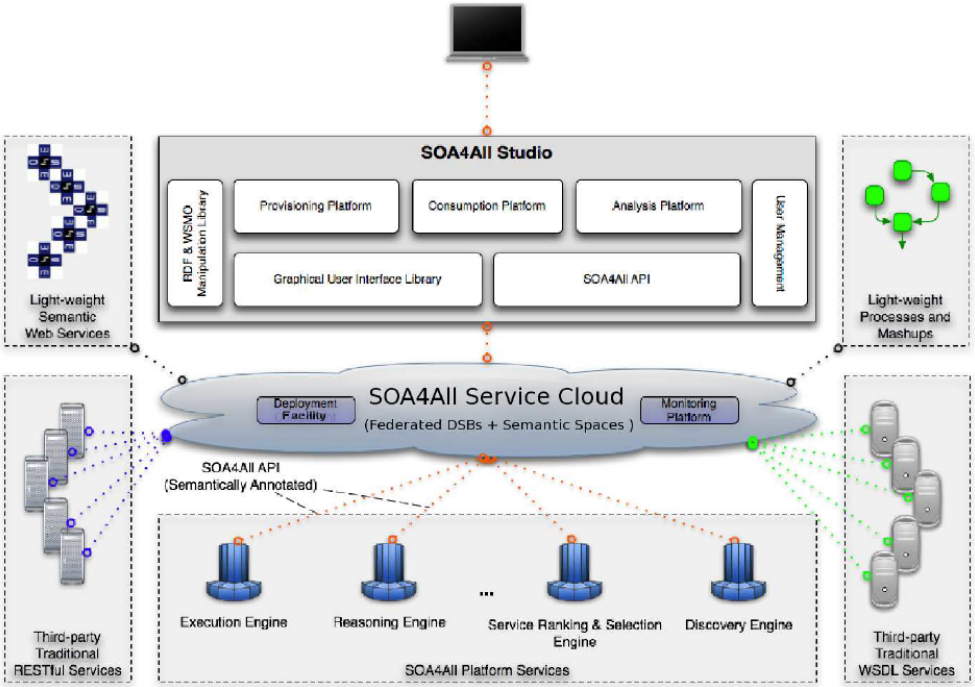

FIGURE 2.39 – L'architecture du projet SOA4ALL. Source : [10].

Comme l'illustre la figure 2.39, la plate-forme SOA4ALL est centrée autour d'un nuage de services auprès duquel sont enregistrés tous les services RESTful et WSDL fournis par des prestateurs externes. Ce nuage est mis en oeuvre par une fédération de bus de services distribués (DSB) PETALS et des espaces sémantiques [77, 64, 10]. La plate-forme SOA4ALL fournit ses propres services comme un moteur d'exécution de services, un moteur de raisonnement sémantique, un moteur de classement et de sélection de services, un moteur de découverte de services, etc. Basé sur les technologies Web 2.0 (HTML5 et JAVASCRIPT), le

63. http://www.inria.fr/equipes/oasis
64. http://cordis.europa.eu/fp7/ict/ssai/docs/fp7call1achievements/soa4all.pdf

studio SOA4All offre une interface Homme - Machine pour permettre à tout un chacun de produire et consommer des services et des processus métiers [31].

Dans ce contexte, la thèse de Virginie Legrand Contes a porté sur la conception d'un moteur d'orchestration décentralisée de services métiers.

2.10.2 La problématique

Une orchestration de services métiers décrit un processus automatique d'organisation, de coordination et de gestion d'un ensemble de services métiers. Cela permet souvent de réaliser un processus métier en composant des services autonomes fournis par différents prestataires métiers. Par exemple, la figure 2.40 illustre un processus métier de réservation de voyages consistant en la réservation d'un billet d'avion, d'une chambre d'hôtel et d'une voiture. Ce processus compose différents services fournis par l'entreprise, une agence de voyages et un loueur de véhicules.

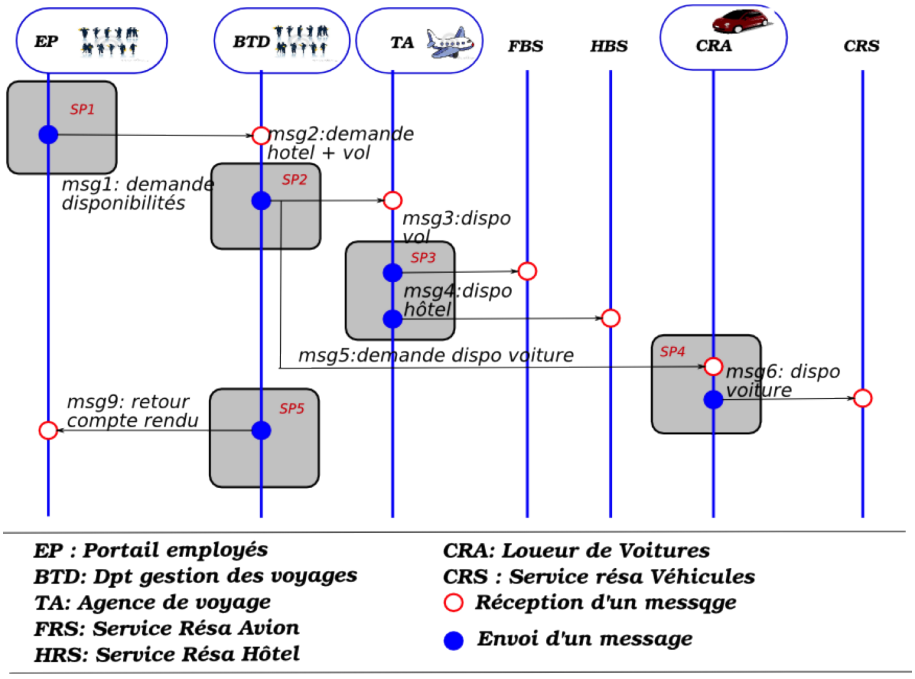

FIGURE 2.40 – Un processus métier de réservation d'un voyage. Source : [87].

Classiquement, même si les services métiers sont distribués sur le réseau, le flot de contrôle d'une orchestration est exécutée sur une seule machine, comme c'est le cas par exemple pour les moteurs WS-BPEL. Cela introduit des problèmes de performance, de sécurité et de maintenance. Le site d'exécution de l'orchestration peut rapidement devenir un goulot d'étrangement. Les données envoyées ou reçues des services doivent toutes transitées sur le réseau. Les services doivent être accessibles depuis n'importe quelle orchestration les utilisant. Les orchestrations s'exécutant sur une longue durée ne peuvent pas être mises à jour sans devoir les arrêter pour effectuer leur mise à jour, perdant ainsi les résultats intermédiaires déjà collectés par l'orchestration.

Pour pallier à ces problèmes, l'objectif de la thèse de Viginie Legrand Contes fut de proposer une exécution décentralisée des orchestrations métiers.

2.10.3 Les contributions

La thèse de Virginie Legrand Contes propose une approche à composants pour l'orchestration de services à large échelle [87]. L'idée initiale est de décomposer un processus métier en un ensemble de sous-processus selon des contraintes spatio-temporelles, comme l'illustre la figure 2.41.

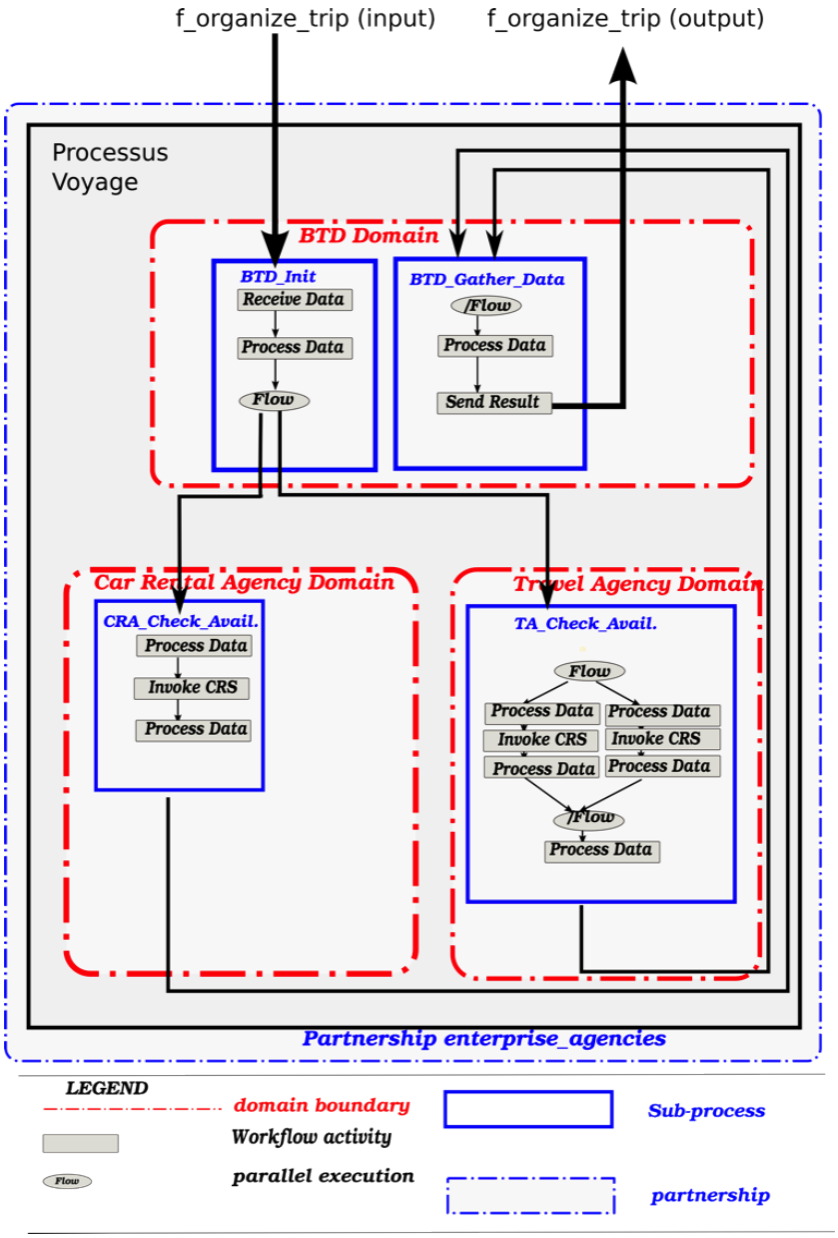

FIGURE 2.41 – Le découpage spatio-temporelle du processus métier de réservation de voyages. Source : [87].

Dans un premier temps, nous identifions les domaines d'exécution du processus métier. Par exemple, le processus `Voyage` est composé de trois domaines : le département gestion des voyages de l'entreprise, l'agence de voyages et le loueur de voitures. Ensuite, le processus mé-

tier est décomposé en sous-processus respectant la localité des données et des services utilisés. Le sous-processus `BTD_Init` contient les instructions recevant les données de la demande de voyage de l'employé et le traitement de celles-ci. Le sous-processus `CRA_Check_Avail` regroupe les instructions du processus invoquant le service de réservation de voitures. Le sous-processus `TA_Check_Avail` regroupe les instructions invoquant les services de l'agence de voyages. Le dernier sous-processus `BTD_Gather_Data` fusionne les résultats et renvoie une réponse. Finalement, les sous-processus sont liés par des dépendances temporelles décrivant dans quelle ordre les sous-processus doivent être exécutés. Ces dépendances peuvent introduire du parallélisme entre les sous-processus comme c'est le cas pour les sous-processus `CRA_Check_Avail` et `BTD_Gather_Data`.

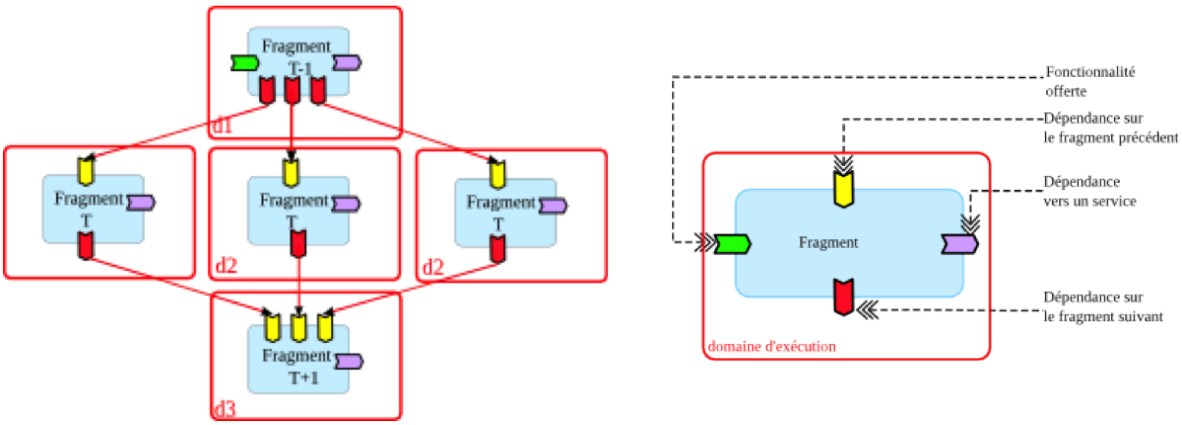

FIGURE 2.42 – Les fragments composant un processus métier. Source : [87].

Chaque sous-processus est alors encapsulé dans un *fragment*. Un fragment est un composant SCA déployé dans un certain domaine d'exécution. Un fragment offre des services exposant les fonctionnalités métiers du sous-processus associé. Un fragment possède des références vers les services métiers invoqués par le sous-processus. Enfin, un fragment possède deux ports pour assembler le fragment avec les autres fragments, comme l'illustre la figure 2.42. Ceux-ci sont des ports orientés données. Le port au dessus du fragment (en jaune) permet de recevoir des données en provenance des fragments précédents. Le port en dessous du fragment (en rouge) permet d'envoyer des données vers les fragments suivants. Finalement, les fragments sont regroupés au sein d'un composite qui matérialise le processus métier global, comme l'illustre la figure 2.43.

Chaque fragment et composite de fragments est équipé de capacités de contrôle réflexif intégrées dans leur membrane sous la forme d'assemblage de composants de contrôle. Ces capacités sont :

- Le **contrôle du déploiement** pour démarrer et stopper le moteur d'orchestration sous-jacent, déployer ou supprimer la définition du processus à exécuter.
- Le **contrôle d'administration** (CX) pour démarrer, suspendre, arrêter l'exécution d'un processus associé au fragment, ajouter ou supprimer des fragments et modifier les liaisons fonctionnelles d'un fragment.
- Le **contrôle de supervision** (S) pour souscrire aux événements survenant durant l'exécution d'un fragment tels que l'invocation / indisponibilité / remplacement d'un service, le déploiement / démarrage / arrêt d'un processus, l'installation / démarrage / arrêt

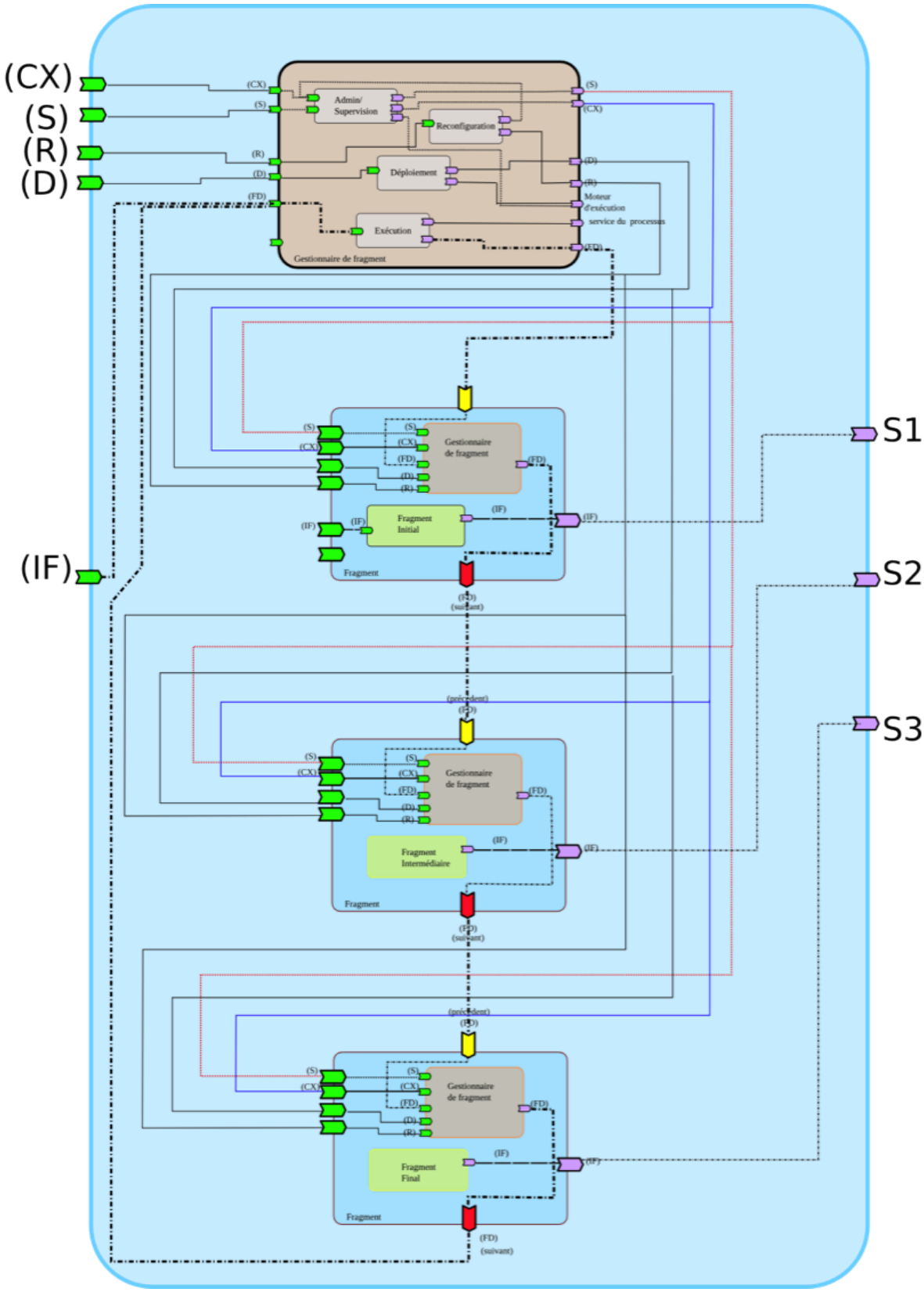

FIGURE 2.43 – La composition de fragments. Source : [87].

d'un moteur d'orchestration, la réception / envoi des données applicatives, la création / ajout / reconfiguration / suppression d'un fragment, etc.
- Le **contrôle de reconfiguration** (R) pour modifier la définition du processus associé au fragment, modifier les références du fragment, etc.
- La **gestion des flux de données inter-fragments** (FD) pour recevoir et émettre des données vers les fragments suivants.

Un prototype de cette approche a été réalisé au dessus de l'intergiciel GCM/ProActive de l'équipe-projet Inria OASIS [11].

2.11 La réflexivité au service de l'évolution des systèmes de systèmes

Cette section résume le contexte, la problématique et les contributions de la thèse de Jonathan Labéjof [80] que j'ai co-encadré avec le Prof. Lionel Seinturier au sein de l'équipe-projet Inria ADAM. Cette thèse CIFRE s'est déroulée au sein de la société Thales Communications & Security (TCS).

2.11.1 Le contexte et la problématique

La société TCS se positionne comme le numéro un européen et parmi les tout premiers mondiaux des systèmes d'information et de communication sécurisés pour les marchés mondiaux de la défense, de la sécurité et du transport terrestre. Ces systèmes sont aussi désignés par l'acronyme C4I (*Command, Control, Computer, Communications and Intelligence*). TCS propose une gamme complète de solutions C4I composées d'équipements, de systèmes logiciels et de services à valeur ajoutée. Toutefois, ses solutions doivent souvent être assemblées avec d'autres solutions provenant de fournisseurs concurrents. Ainsi, TCS est souvent amené à construire des *systèmes de systèmes*.

Un système de systèmes, ou SoS, est une collection de systèmes autonomes, dédiés ou orientés tâche qui concentrent ensemble leurs ressources et leurs capacités afin de créer un nouveau système qui offre plus de fonctionnalités et de performances que la somme des fonctionnalités issues des systèmes constituants [69].

Un exemple de système de systèmes commercialisé par Thales est le système TACTICOS pour superviser et contrôler des infrastructures et du matériel naval afin d'effectuer des missions militaires. Comme l'illustre la figure 2.44, chaque navire est un système de systèmes autonomes. L'ensemble de la flotte est alors un système de systèmes hétérogènes mais devant interopérer. Un sous-ensemble de TACTICOS a été utilisé pour évaluer les contributions de la thèse de Jonathan Labéjof.

Afin d'assurer l'autonomie de chaque système, un système de systèmes met souvent en oeuvre des interactions asynchrones via des intergiciels orientés messages (ou MOM). L'article [42] présente en détails les différentes facettes de ce type d'intergiciels. Par exemple, les interactions peuvent être acheminées selon le sujet d'intérêt, e.g., la météo actuelle, dans les intergiciels de type *Publish/Subscribe* ou transportées par des queues de messages, e.g., le plan de vol de l'avion BX035. Il existe une pléthore d'intergiciels asynchrones : certains sont propriétaires comme 0MQ ou KryoNet tandis que d'autres respectent des standards tels que

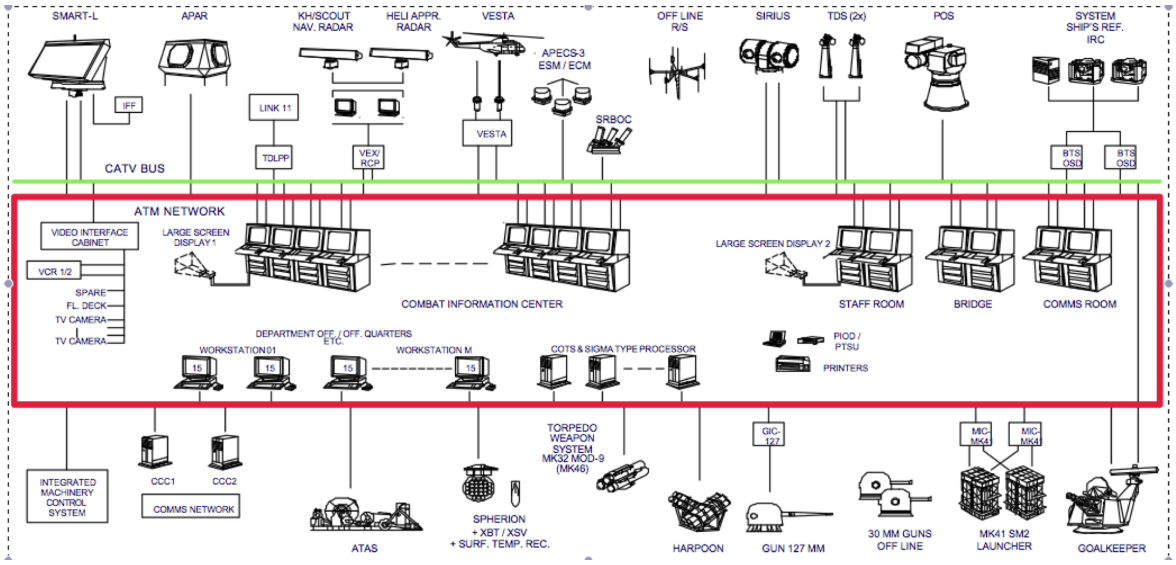

FIGURE 2.44 – Le système de systèmes TACTICOS. Source : [80].

JMS, AMQP [65], ou OMG Data Distribution Service (DDS) [161]. Chaque système peut utiliser un MOM différent en fonction de ses besoins et contraintes. Cette hétérogénéité pose alors de sérieux problèmes d'**interopérabilité entre intergiciels asynchrones** dans le contexte des systèmes de systèmes.

Un système de systèmes est amené à évoluer au fil du temps afin de prendre en compte de nouveaux sous-systèmes, des changements de mission ou technologiques. Ces évolutions doivent souvent être effectuées à chaud, c'est-à-dire sans nécessité d'arrêter tous les sous-systèmes pour effectuer un changement sur un système donné.

Pour résoudre ces différents problèmes, la thèse de Jonathan Labéjof propose d'appliquer la réflexivité [217] au service de l'évolution des systèmes de systèmes [80].

2.11.2 Les contributions

Comme l'illustre la figure 2.45, la thèse de Jonathan Labéjof propose l'approche nommée R-* constitué de trois contributions R-DDS, R-MOM et R-EMS.

- **R-DDS** adresse la reconfigurabilité d'intergiciel Data Distribution Service (DDS) aussi bien au niveau métier que extra-fonctionnel.
- **R-MOM** adresse l'interopérabilité entre intergiciels asynchrones (MOM) aussi bien au niveau des protocoles réseaux que des personnalités applicatives (API).
- **R-EMS** adresse la spécification de systèmes de systèmes évolutifs.

2.11.2.1 R-DDS : la reconfigurabilité de l'intergiciel DDS

La spécification OMG DDS définit un intergiciel de distribution de données pour systèmes temps-réels et embarqués (RT-E). Ce standard est largement utilisé dans les domaines de la défense, de l'aérospatial, du naval et du ferroviaire par des sociétés comme THALES, EADS, etc. Cet intergiciel propose une interface de type *Publish/Subscribe* orientée données. Les

65. *Advanced Message Queue Protocol* - `http://www.amqp.org/`

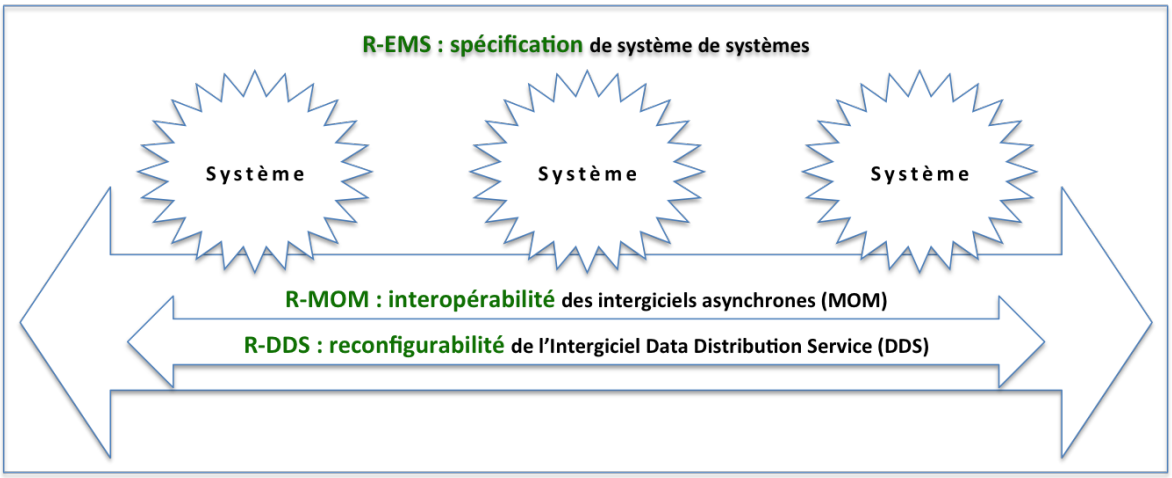

FIGURE 2.45 – L'approche R-* comprenant les contributions R-DDS, R-MOM et R-EMS.

messages échangés sont structurés et définis par des types de données. Ces types sont les sujets d'intérêt pour les publications et les souscriptions. L'interface OMG DDS fournit des opérations de lecture/écriture de données. OMG DDS définit 60 concepts et 23 qualités de service (QoS) pour la distribution de données. Pour plus de détails sur l'interface OMG DDS, se reporter à la thèse [80] ou plus exhaustivement à la spécification [161]. Il existe aujourd'hui plus d'une dizaine d'implantations de la spécification OMG DDS telles que OPENSPLICE de PRISMTECH et RTI CONNEXT DDS PROFESSIONAL.

Cependant, l'interface de programmation OMG DDS offre des capacités de reconfiguration très limitées. Il est rarement possible de modifier les propriétés d'un objet DDS a posteriori. Ici, R-DDS permet de reconfigurer toutes les propriétés de tous les constructions OMG DDS durant l'exécution, comme par exemple choisir l'implantation DDS à mettre en oeuvre, ajouter / modifier / supprimer des traitements dynamiques de QoS, changer le type de données associé à une entité de lecture / écriture, etc.

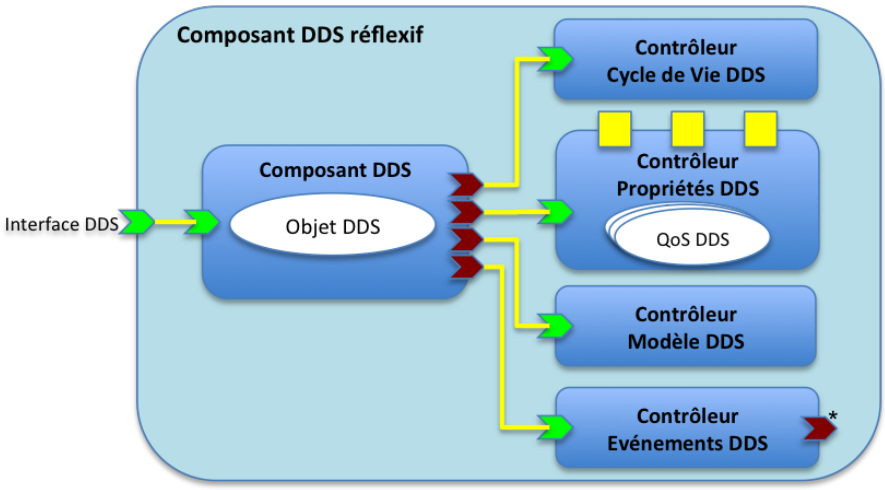

FIGURE 2.46 – Le patron de construction des composants R-DDS.

Pour cela, chaque entité DDS est réifiée sous la forme d'un composite FraSCAti réflexif comme l'illustre la figure 2.46. Ce composite offre un service pour l'interface DDS associée à l'entité DDS réifié. Ce composite est composé de cinq composants :

- le **composant DDS** encapsule l'objet DDS réifié. Les appels de service sont interceptés, si nécessaire délégués à l'un des quatre autres composants, puis envoyés finalement à l'objet DDS. L'implantation de ce composant est spécifique à chaque concept DDS.
- Le **contrôleur de cycle de vie DDS** permet de suspendre temporairement le composant DDS afin d'exécuter des reconfigurations puis de le redémarrer plus tard.
- Le **contrôleur de propriétés DDS** stocke les propriétés DDS de l'objet encapsulé et en autres les qualités de service DDS. Ce contrôleur permet de ajouter / modifier / supprimer des qualités de service.
- Le **contrôleur du modèle DDS** gère les relations de composition entre composites DDS.
- Le **contrôleur d'événements DDS** gère les *listeners* abonnés aux événements générés par l'objet DDS réifié.

Ce patron de construction a été appliqué à tous les concepts de la spécification OMG DDS et a permis de produire 60 composites FraSCAti. L'implantation de R-DDS supporte aussi bien le produit OpenSplice que le produit RTI.

La figure 2.47 présente l'évaluation du temps moyen pour 50 envois / réceptions de 10 000 données de différentes tailles (64, 512, 1024 et 32768 octets). Le surcoût moyen observé est de l'ordre de 7% du à la réification en composites des entités DDS. Toutefois, ce surcoût reste compatible avec les exigences temps-réels du système TACTICOS. Le bénéfice est que le système devient reconfigurable à chaud.

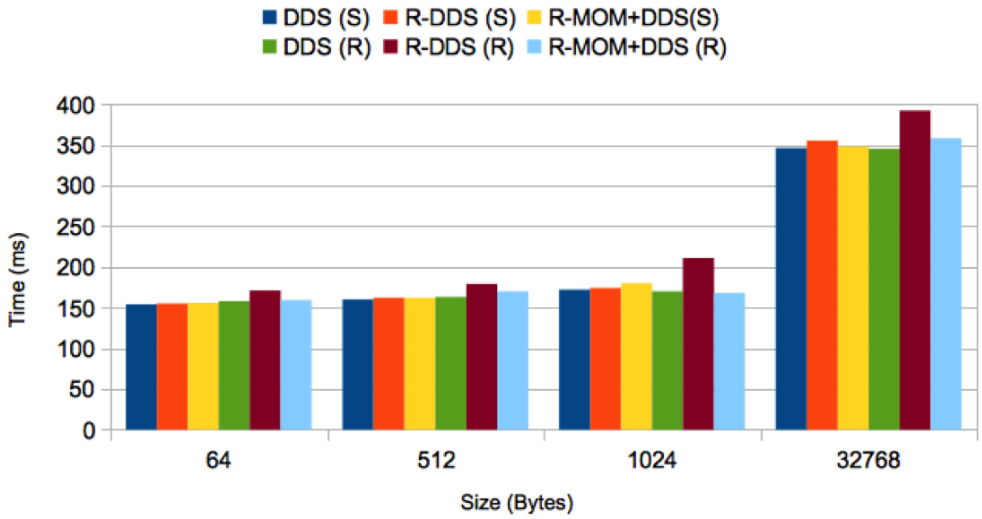

Figure 2.47 – Comparaison des temps moyens de traitement entre DDS, R-DDS et R-MOM DDS. Source : [80].

La contribution R-DDS a fait l'objet d'un dépôt de brevet en Europe et aux Etats-Unis [79].

2.11.2.2 R-MOM : l'interopérabilité des intergiciels asynchrones

La deuxième contribution R-MOM s'attaque à l'interopérabilité reconfigurable entre intergiciels asynchrones. Pour cela, R-MOM permet de construire des passerelles d'interopérabilité entre intergiciels asynchrones. Par exemple, cela permet de recevoir des données d'un système temps-réel via DDS et de les publier vers un système d'information via JMS comme l'illustre la figure 2.49 ou bien d'appliquer des qualités de services DDS sur des données JMS. Ainsi, R-MOM peut être vu comme un intergiciel d'intergiciels car il intègre différents intergiciels asynchrones existants et il offre une interface unique pour construire des passerelles d'interopérabilité.

R-MOM est un canevas à composants comparable au canevas Dream [86, 194]. La différence est que Dream vise la construction d'intergiciels asynchrones tandis que R-MOM se focalise sur la construction de passerelles d'interopérabilité entre intergiciels asynchrones existants. Ces deux canevas sont ainsi complémentaires.

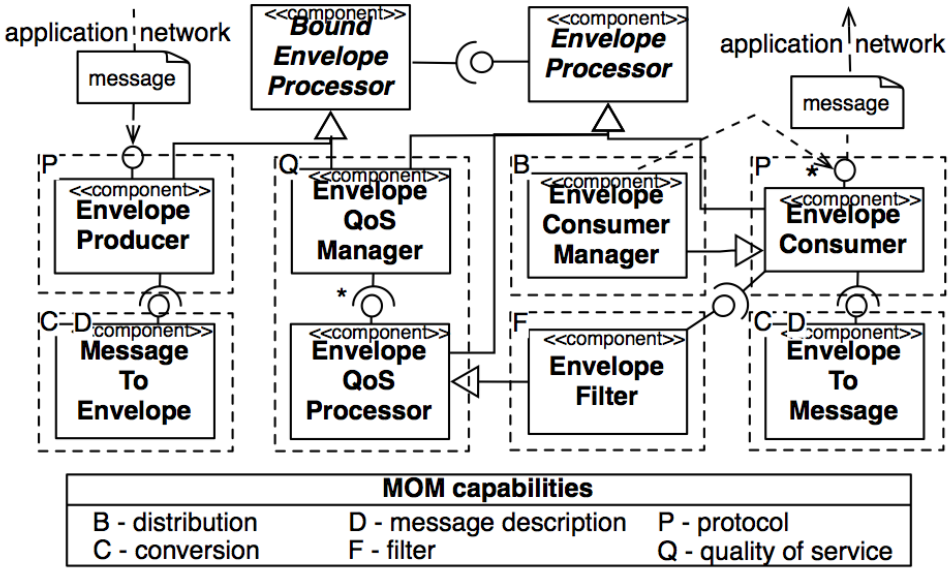

Figure 2.48 – Le diagramme des composants R-MOM. Source : [78, 80].

La figure 2.48 montre les différents composants R-MOM ainsi que leurs relations. Ces composants s'échangent des *enveloppes*. `Enveloppe` est la structure de données représentant un message au sein de R-MOM. Cette structure de données est agnostique par rapport aux intergiciels asynchrones existants et auto-descriptive, c'est-à-dire contient des méta-données sur le message. R-MOM propose huit types de composants concrets :

- **Envelope Producer** a pour rôle de recevoir des messages depuis un intergiciel asynchrone puis d'en produire des enveloppes.
- **Message to Envelope** a pour rôle de convertir un message en une enveloppe.
- **Envelope QoS Manager** a pour rôle d'appliquer un ensemble de qualités de service sur des enveloppes.
- **Envelope QoS Processor** a pour rôle d'appliquer une certaine qualité de service sur des enveloppes.

- **Envelope Consumer Manager** a pour rôle de router des enveloppes vers plusieurs consommateurs d'enveloppe.
- **Envelope Consumer** a pour rôle de consommer des enveloppes pour produire des messages vers un intergiciel asynchrone.
- **Envelope Filter** a pour rôle de filtrer des enveloppes.
- **Envelope to Message** a pour rôle de convertir une enveloppe en un message.

Cet ensemble de composants élémentaires peut alors être composé " à la carte " pour produire différents chaînes d'interopérabilité entre intergiciels asynchrones. Des composants peuvent être partagés entre les différentes chaînes d'interopérabilité.

La figure 2.49 illustre une passerelle d'interopérabilité entre DDS et JMS. Des données DDS sont reçues par le composant `Producteur Enveloppe`. Celui-ci demande au composant `Convertisseur M2E` de convertir les données DDS en enveloppes R-MOM puis transmet ces enveloppes au composant `Gestionnaire QoS`. Celui-ci invoque chacun de ces composants `Traitement de QoS`. L'enveloppe résultante est alors envoyée au composant `Distributeur` qui route celle-ci vers différents composants `Consommateur Enveloppe`. Le consommateur peut filtrer les enveloppes via les composants `Filtre`, puis convertit les enveloppes en messages JMS et les envoie vers l'intergiciel JMS.

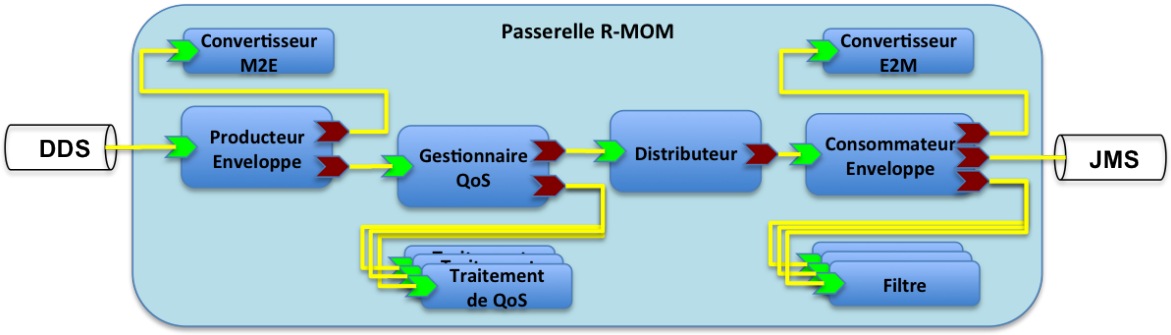

FIGURE 2.49 – Une passerelle R-MOM d'interopérabilité de DDS vers JMS.

Construit au dessus de FRASCATI, R-MOM supporte l'interopérabilité entre onze moyens de communication asynchrone : des protocoles Internet tels que UDP et TCP, l'intergiciel JGROUPS, des intergiciels JMS tels que JBOSSMQ, JORAM, ACTIVEMQ et OPENJMS, les intergiciels AMQP tel que RABBITMQ, des intergiciels DDS tel que OPENSPLICE et des intergiciels propriétaires tels que 0MQ et KRYONET. L'intégration d'un nouvel intergiciel asynchrone consiste à développer quatre implantations de composants, c'est-dire le producteur et le consommateur d'enveloppes ainsi que les deux convertisseurs. R-MOM supporte six qualités de service définies par DDS dont trois sont aussi définies par JMS : durabilité, ordre de réception, partage, durée de vie, historique et priorité.

La contribution R-MOM a fait l'objet d'une publication dans un atelier de travail de la conférence internationale EDOC [78].

2.11.2.3 R-EMS : la spécification de systèmes de systèmes

La troisième contribution R-EMS s'intéresse à la spécification de systèmes de systèmes. Plus particulièrement, R-EMS permet de décrire la configuration d'un système de systèmes, d'automatiser le déploiement de celui-ci, puis de le reconfigurer. R-EMS se présente sous

la forme d'un méta-modèle EMS équipé d'une syntaxe concrète XText. Ce méta-modèle contient des constructions pour décrire la structure d'un système de systèmes et des constructions pour programmer la dynamique du système de systèmes. Les modèles R-EMS sont interprétés à l'exécution. R-EMS peut ainsi être vu comme une approche Models@run.time [14]. Pour plus de détails, le lecteur peut se reporter à la thèse [80].

2.12 La plate-forme multi-nuages soCloud

Cette section résume le contexte, la problématique et les contributions de la thèse de Fawaz Paraiso [174] que j'ai co-encadré avec le Prof. Lionel Seinturier au sein de l'équipe-projet Inria ADAM puis Spirals. Cette thèse propose soCloud : une plate-forme multi-nuages distribuée pour la conception, le déploiement et l'exécution d'applications distribuées à large échelle.

2.12.1 Le contexte et la problématique

Le 25 août 2006, la société Amazon annonçait le lancement de son service Elastic Compute Cloud (EC2) permettant à des tiers de louer de la puissance de calcul de ses serveurs sous-utilisés. Depuis, l'informatique en nuage, ou *Cloud Computing*, a connu un essor considérable aussi bien dans l'industrie en général [21] que dans le milieu de la recherche, comme en témoigne un récent article publié dans *IEEE Transactions on Cloud Computing* [65] présentant une analyse scientométrique de la littérature portant sur l'informatique en nuage.

En septembre 2011, le National Institute of Standards and Technology (NIST) publia sa définition [66] de l'informatique en nuage [110] : "*Cloud computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*". Le NIST identifia cinq caractéristiques essentielles de l'informatique en nuage :

1. **À la demande en libre-service**. Les ressources informatiques (processeurs, disques, réseaux, serveurs, services, applications, etc.) sont obtenues via un procédé automatisé sans intervention humaine.
2. **Accessible par le réseau**. Les ressources sont accessibles par des protocoles réseaux standardisés depuis une hétérogénéité de clients.
3. **Pool de ressources**. Les ressources sont partagées entre les utilisateurs, virtualisées et délocalisées.
4. **Élasticité rapide**. L'utilisateur a l'impression que les ressources sont illimitées, rapidement disponibles et automatiquement.
5. **Service sur mesure**. Les ressources sont dimensionnées en fonction des besoins et sont payés à l'usage.

Le NIST identifia trois principaux modèles de prestation de l'informatique en nuage, comme l'illustre la figure 2.50 :

1. **Infrastructure as a Service** (IaaS) inclut les ressources matérielles telles que les processeurs, le stockage et le réseau.

66. Même si d'autres définitions existent, la définition du NIST fait référence.

2. **Platform as a Service** (PaaS) inclut les environnements d'exécution, les intergiciels, les systèmes de fichiers, les bases de données, etc.
3. **Software as a Service** (SaaS) inclut les applications à destination des usagers finaux comme le courrier électronique, les outils collaboratifs, les sites de partage de vidéos / musiques, les réseaux sociaux, etc.

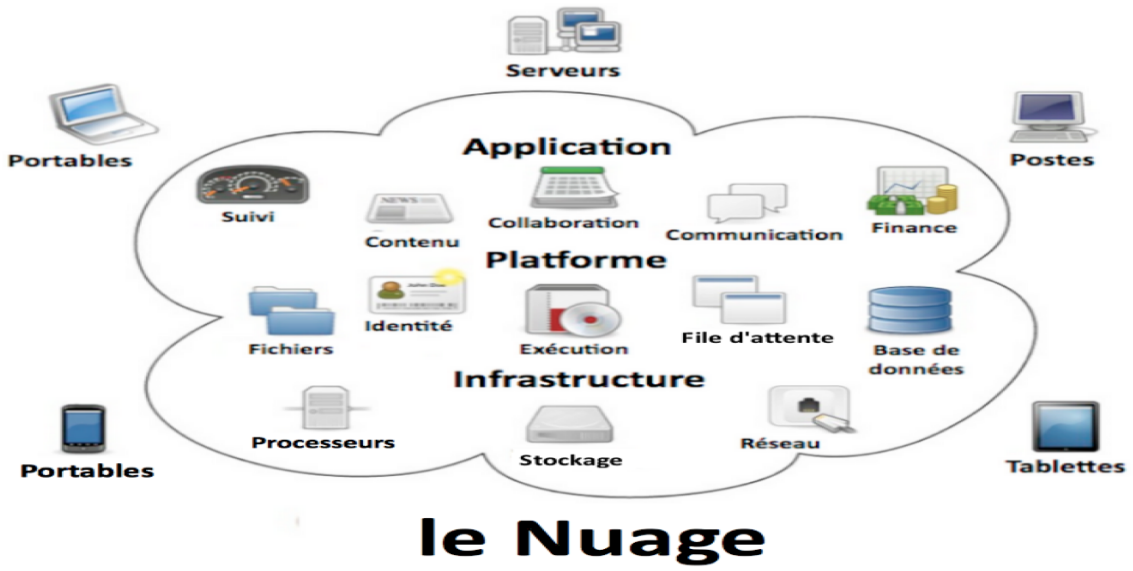

FIGURE 2.50 – L'informatique en nuage. Source : Wikipédia.

Enfin, le NIST identifia quatre modèles de déploiement de l'informatique en nuage : les nuages publics, privés, hybrides et communautaires.

Aujourd'hui, il existe une pléthore d'offres d'informatique en nuage telles que les offres IaaS d'AMAZON, de GOOGLE, d'IBM, de HP et les offres PaaS comme CLOUDBEES, CLOUD FOUNDRY, DOTCLOUD, GOOGLE APP ENGINE, HEROKU, JELASTIC, OPENSHIFT, pour n'en citer que quelques unes. Ainsi, l'utilisateur est face à un premier problème : quelle offre choisir ? Une fois une offre choisie, l'utilisateur se retrouve pris en otage par cette offre (aussi appelé les *vendor lock-ins*) car la portabilité des applications et des données n'est pas assurée entre clouds [186, 189]. Ainsi, migrer une application d'un nuage à un autre est difficile voire impossible. De même, construire une application s'exécutant simultanément dans plusieurs nuages se heurte aux problèmes de portabilité et d'interopérabilité entre nuages.

Pour cela, l'informatique multi-nuages propose de s'intéresser à l'informatique en multiples nuages [174]. [61] et [187] proposent deux taxonomies de l'informatique multi-nuages dans lesquelles apparaissent trois grandes catégories de solutions : 1) les fédérations de nuage telle que les projets européens CONTRAIL [22] et RESERVOIR [196], 2) les inter-nuages (ou *Inter-Clouds*) telles que les projets COMPATIBLEONE [242], INTERCLOUD [20] et STRATOS [181], 3) les multi-nuages (ou *Multi-Clouds*) telles que les projets européens 4CAASST [52], CLOUD4SOA [70], MODACLOUDS [139], MOSAIC [188, 141], PAASAGE [166] et notre projet SOCLOUD [174]. Les deux premières catégories s'attaquent à la problématique de l'interopérabilité entre nuages. La dernière catégorie fait face à quatre autres problématiques majeures : la **portabilité** multi-nuages, l'**approvisionnement** multi-nuages, l'**élasticité** multi-nuages et la **haute disponibilité** multi-nuages comme nous l'avons identifiées dans [178].

La thèse de Fawaz Paraiso [174] a alors porté sur ces quatre problématiques et sur deux questions de recherche en informatique multi-nuages :

- **QR1** : Quel modèle pour des applications multi-nuages ?
- **QR2** : Quelle plate-forme pour des applications multi-nuages ?

2.12.2 Les contributions

Comme l'illustre la figure 2.51, la thèse de Fawaz Paraiso [174] propose un modèle pour concevoir et développer des applications distribuées multi-nuages (voir la section 2.12.2.1) et une plate-forme pour déployer et exécuter ces applications dans un environnement multi-nuages (voir la section 2.12.2.2). Ce modèle et cette plate-forme SOCLOUD ont été validés sur trois applications distribuées à large échelle (voir la section 2.12.2.3).

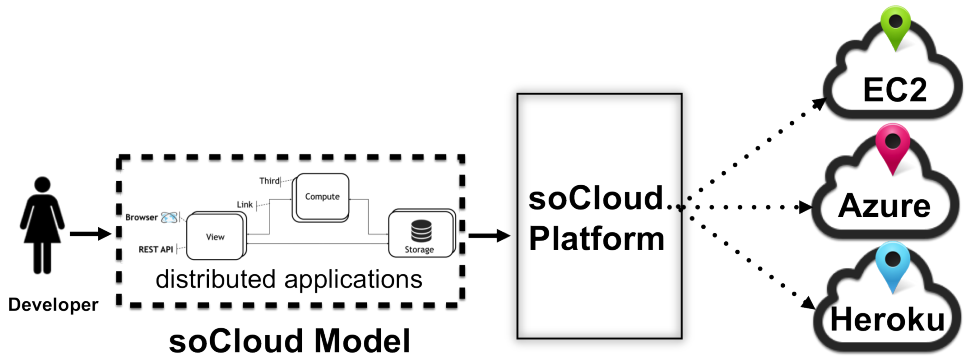

FIGURE 2.51 – Le modèle et la plate-forme SOCLOUD.

2.12.2.1 Le modèle soCloud

Le modèle d'applications multi-nuages SOCLOUD est une extension du modèle d'applications FRASCATI et bénéficie ainsi des capacités réflexives de FRASCATI [134, 175]. Comme l'illustre la figure 2.52, une application SOCLOUD est modélisée par un composite SCA et est conditionné sous la forme d'une contribution SCA (c'est-à-dire un fichier ZIP). Chacun composante (ou tier) de l'application multi-nuages est modélisée par un composant SCA. La première extension du modèle SOCLOUD est que l'implantation des composants d'une application SOCLOUD est réalisée par une contribution SCA. La seconde extension du modèle SOCLOUD est que les composants d'une application SOCLOUD sont annotés avec des méta-informations prises en charge par la plate-forme SOCLOUD.

Comme l'illustre la figure 2.53, SOCLOUD fournit quatre catégories d'annotations / contraintes de placement, d'exécution, de disponibilité et d'élasticité :

- L'annotation **location** permet de placer un composant à une certaine localisation comme par exemple un continent, un pays, une ville et/ou un fournisseurs de nuages. En l'absence d'une telle annotation, la plate-forme SOCLOUD choisit automatiquement une localisation respectant les autres contraintes.
- L'annotation **closer** permet de placer un composant à proximité d'une autre composant. Cela permet de réduire la latence entre des composants interagissant fortement. Dans la figure 2.52, le composant `Métier` est placé à proximité du composant `Stockage`.

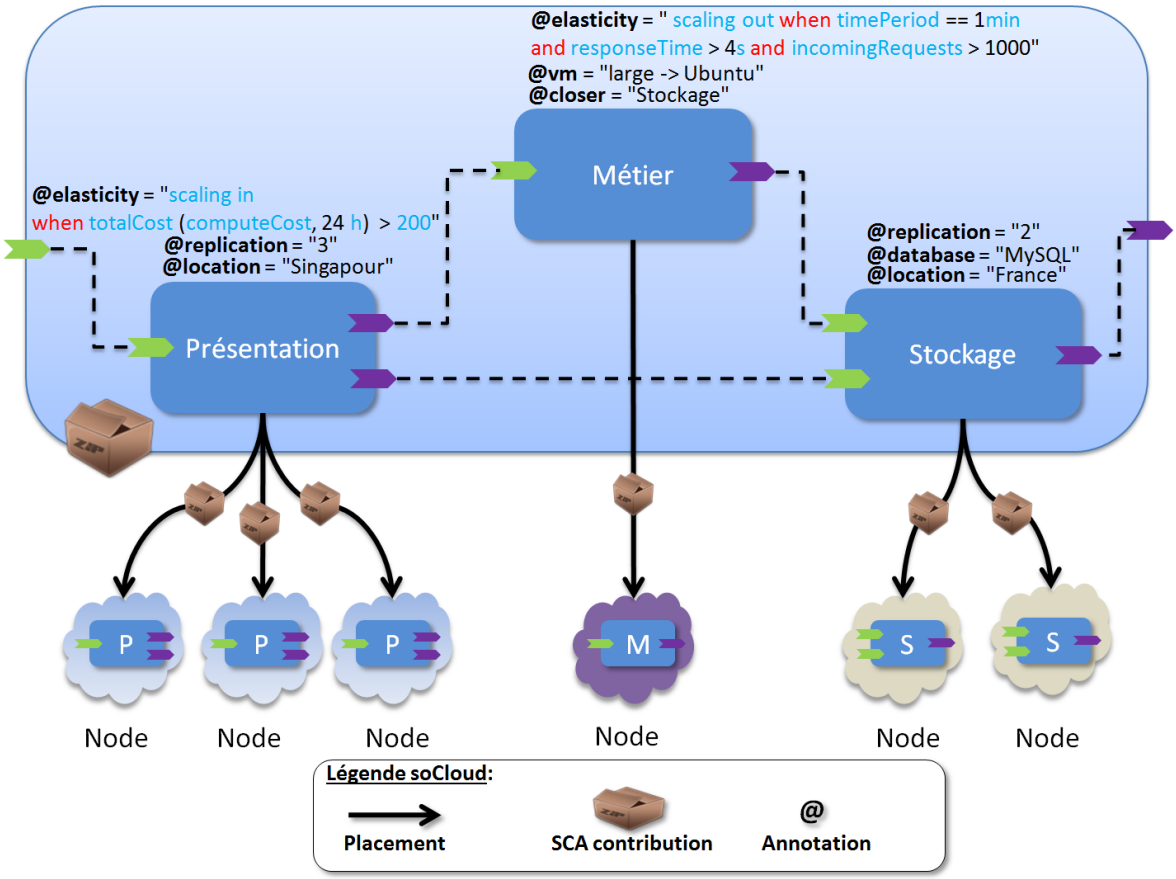

FIGURE 2.52 – Une application SOCLOUD. Source : [174].

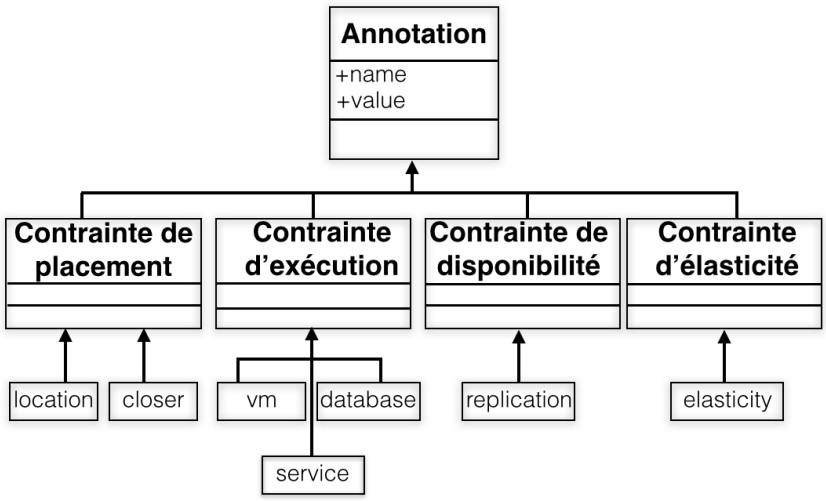

FIGURE 2.53 – Les annotations SOCLOUD. Source : [174].

- L'annotation **vm** permet de choisir la nature de la machine virtuelle sur laquelle s'exécutera le composant. Cela inclut le dimensionnement de la machine virtuelle (e.g., `micro`, `small`, `medium`, `large`, `xlarge`) et le type de l'image système (e.g., `Ubuntu`, `Windows`, etc.).
- L'annotation **service** permet d'indiquer que le composant nécessite un certain service offert par le nuage sur lequel il s'exécutera.
- L'annotation **database** permet d'indiquer que le composant nécessité une certaine base de données.
- L'annotation **replication** permet d'indiquer le nombre minimal de répliquas d'un composant.
- L'annotation **elasticity** permet d'indiquer des règles d'élasticité pour ce composant. Pour cela, un langage dédié est proposé dans [174].

2.12.2.2 La plate-forme soCloud

La plate-forme soCloud est en charge de déployer les applications soCloud, de les héberger dans des conteneurs FraSCAti, de les monitorer pour assurer leur élasticité et leur haute disponibilité [177]. Pour cela, la plate-forme soCloud met en place une boucle de contrôle autonomique de type MAPE-K (*Monitoring, Analyse, Planification, Execution and Knowledge*) [71].

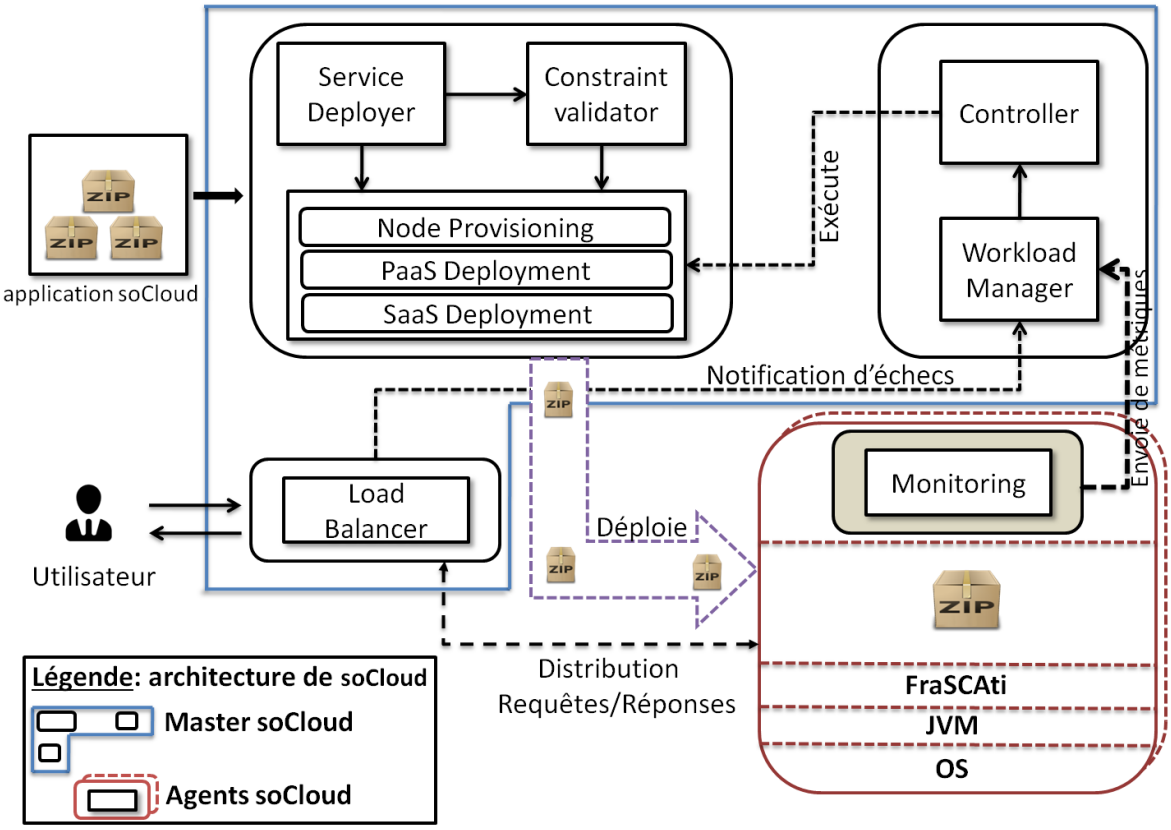

FIGURE 2.54 – L'architecture à composants de la plate-forme soCloud. Source : [174].

Comme l'illustre la figure 2.54, l'architecture à composants de la plate-forme soCloud est constituée de deux parties : les **Masters** et les **Agents**.

Un master soCloud prend en charge le déploiement des applications soCloud (composant `Service Deployer`), c'est-à-dire la validation des annotations (composant `Constraint Validator`), l'approvisionnement de machines virtuelles (composant `Node Provisioning`), le déploiement des agents soCloud (composant `PaaS Deployment`) et le déploiement des composants constituant les applications soCloud (composant `SaaS Deployment`). Cet ensemble de composants constitue les parties exécution et connaissance de la boucle MAPE-K.

L'agent soCloud est le conteneur pour déployer et héberger les composants des applications soCloud. Il inclut la plate-forme d'exécution FraSCAti ainsi que le monitoring des composants soCloud. Ce composant de monitoring notifie le composant `Workload Manager` du master soCloud. Celui-ci analyse les événements de monitoring en fonction des règles d'élasticité associées aux composants d'une application soCloud. La planification est réalisée par le composant `controller` qui invoque les services de déploiement du master.

Le master prend aussi en charge la répartition de charge entre les répliquas d'un même composant soCloud via le composant `Load Balancer`. En cas de défaillance ou inaccessibilité d'un composant soCloud alors le répartiteur de charge notifie le composant `Workload Manager`, ce qui entraine un redéploiement du composant en fonction des règles d'élasticité et du nombre de répliquas minimum de chaque composant.

Afin d'assurer une haute disponibilité de la plate-forme soCloud, les masters et les agents sont répliqués sur plusieurs nuages indépendants. Pour cela, la plate-forme soCloud a été portée sur dix nuages aussi bien des IaaS (Amazon EC2, Windows Azure, DELL KACE, Eucalyptus) que sur des PaaS (Appfog, CloudBees, dotCloud, Heroku, OpenShift, Jelastic), comme l'illustre la figure 2.55. A notre connaissance, soCloud est la plate-forme multi-nuages supportant le plus grand nombre de nuages distincts [134].

Pour plus de détails sur la mise en oeuvre et l'implantation de la plate-forme soCloud, le lecteur pourra se reporter à notre article dans la revue Computing [178] ou plus exhaustivement à la thèse de Fawaz Paraiso [174].

2.12.2.3 Trois applications soCloud

Le modèle et la plate-forme soCloud ont été mis en oeuvre sur trois applications multi-nuages à large échelle : la plate-forme APISENSE, la plate-forme DiCEPE et une application de surveillance de réseau pair-à-pair.

APISENSE est une plate-forme répartie pour la conception, le déploiement et l'exécution de campagnes de collecte de données sur des terminaux intelligents, proposée dans le cadre de la thèse de Nicolas Haderer [62]. APISENSE met en oeuvre un ensemble de serveurs déployés dans les nuages (`Sensing Node`, `Central Node` et `Sensing Storage`), comme l'illustre la figure 2.56. Cette partie serveur d'APISENSE a été construite comme une application soCloud, comme discuté dans [63].

Dans [176], nous avons proposé DiCEPE une plateforme distribuée de traitement d'événements complexes (*Complex Event Processing*). Comme l'illustre la figure 2.57, chaque noeud DiCEPE est un assemblage de composants FraSCAti constitué d'un composant `Engine` encapsulant le moteur de traitement d'événements complexes, de composants `Statement` encapsulant une requête sur le flux d'événements, de composants `Listener` recevant les événements filtrés par les requêtes. Les noeuds DiCEPE peuvent être interconnectés via différentes liai-

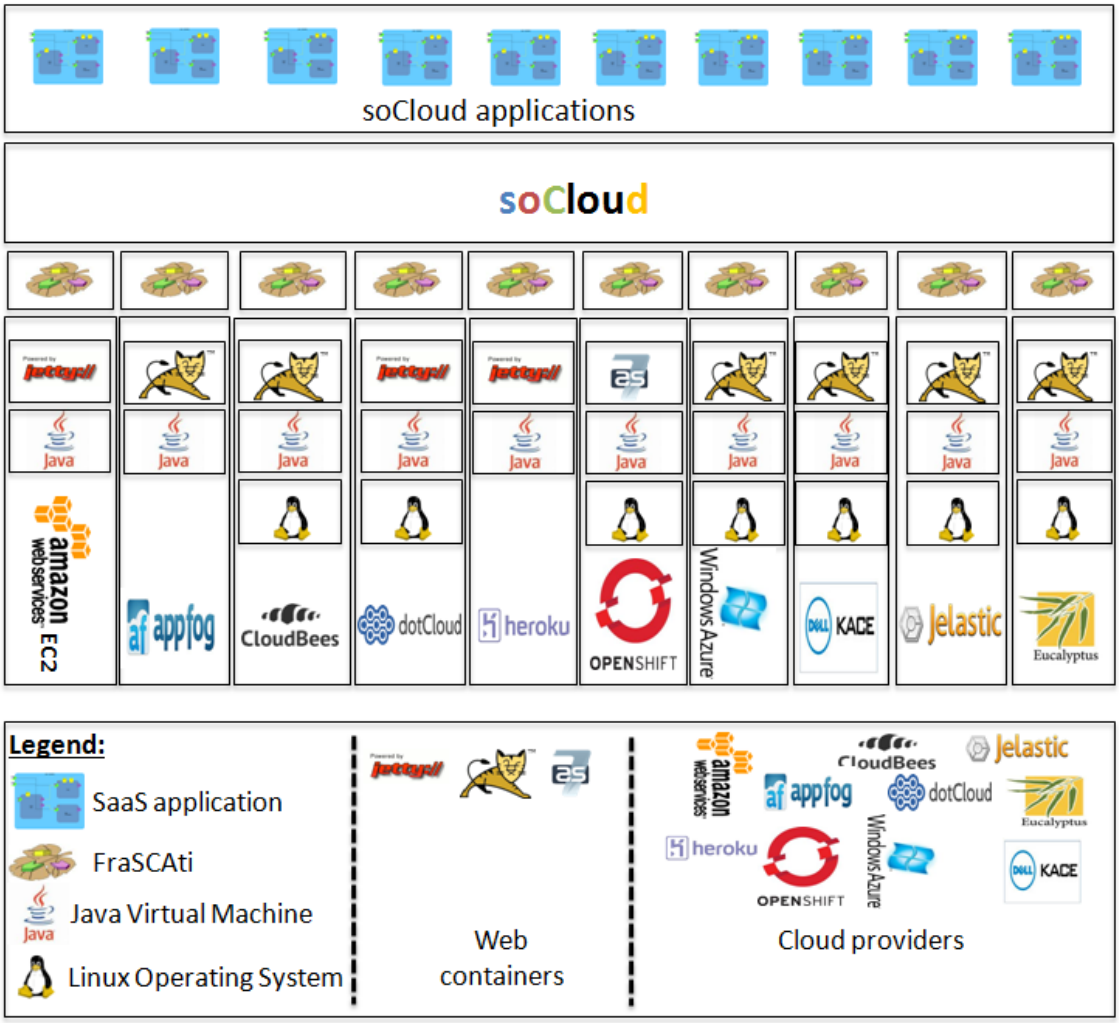

FIGURE 2.55 – Le déploiement de SOCLOUD dans les nuages. Source : [174].

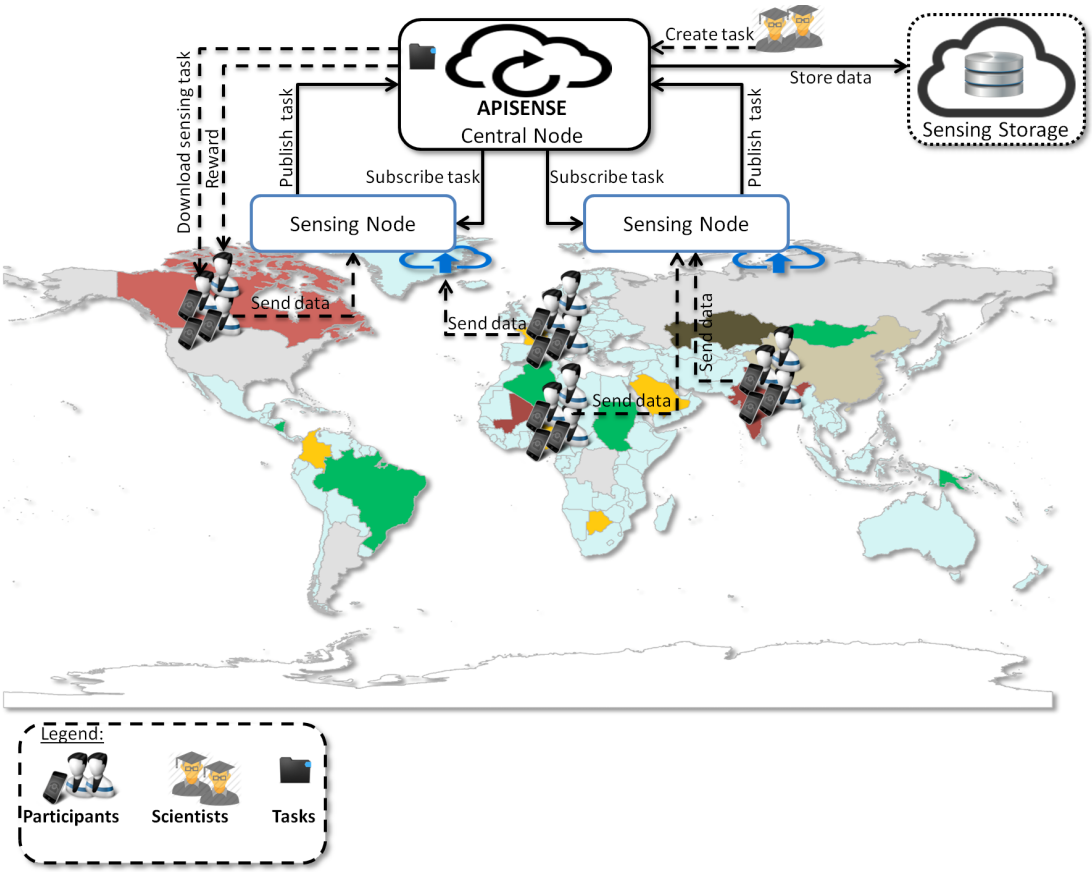

FIGURE 2.56 – La plate-forme APISENSE. Source : [174].

sons d'interopérabilité (REST, JMS, etc.) et sont déployés dans les nuages via la plate-forme soCloud.

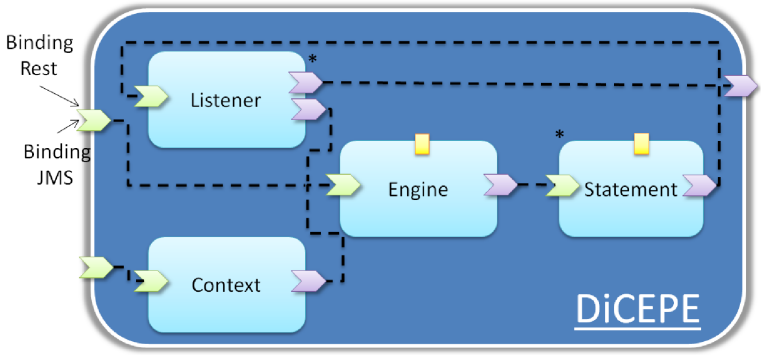

Figure 2.57 – La plate-forme DiCEPE. Source : [174].

Dans [134], nous avons proposé une application de surveillance du réseau entre les nuages. Dans chaque nuage, nous déployons un pair de surveillance et ce pair est connecté avec tous les autres pairs. Comme l'illustre la figure 2.58, lorsque l'utilisateur interroge un pair alors celui-ci initie une communication avec chacun des autres pairs et mesure la latence de cette communication. Sur la figure 2.58, les traits en couleur représentent la latence entre un pair déployé dans le nuage Jelastic localisé en Allemagne et les autres pairs répartis dans divers nuages à travers le monde. Plus le trait est grand, plus la latence est faible. On constate ainsi que la latence peut fortement varier d'un nuage américain à l'autre.

FIGURE 2.58 – L'application de monitoring pair à pair dans les nuages. Source : [134].

Chapitre 3

Bilan et perspectives

Sommaire

3.1 Bilan **87**
3.2 Perspectives **88**

Ce chapitre dresse le bilan synthétique de mes activités de recherche depuis 1997 et présente les grandes lignes de mon projet de recherche pour les années à venir.

3.1 Bilan

Depuis l'obtention de ma thèse de doctorat en 1997, j'ai co-encadré huit thèses de doctorat dont R. Marvie [93], M. Vadet [225], R. Rouvoy [199], J. Dubus [34], A. Plšek [190], V. Legrand Contes [87], J. Labéjof [80], F. Paraiso [174] et j'ai encadré sept mémoires de Master dont R. Marvie [92], S. Leblanc [81], M. Vadet [224], R. Rouvoy [198], F. Briclet [17], J. Dubus [33] et A. Tiberghien [222].

Mes travaux de recherche s'inscrivent dans le domaine de l'intergiciel et ont porté principalement sur :

- le scriptage d'objets répartis CORBA via le langage CorbaScript,
- la séparation des préoccupations dans les architectures logicielles à base de composants (thèse de R. Marvie [93]),
- les conteneurs ouverts pour composants CORBA (mémoire puis thèse de M. Vadet [224, 225]),
- les contrats de courtage CORBA (mémoire de S. Leblanc [81]),
- le déploiement réparti de composants CORBA (mémoires de F. Briclet [17] et J. Dubus [33] puis thèse de J. Dubus [34]),
- la gestion des transactions dans les intergiciels (mémoire et thèse de R. Rouvoy [198, 199]),
- la programmation par annotations de composants logiciels (thèse de R. Rouvoy [199]),
- le déploiement de systèmes distribués hétérogènes (thèse de J. Dubus [34]),
- les composants Java temps-réels (thèse de A. Plšek [190]),
- la formalisation du modèle de composants Fractal [136],
- l'adaptabilité des architectures orientées services via FraSCAti (*cf.* Chapitre 2),

- l'orchestration de services à large échelle (thèse de V. Legrand Contes [87]),
- l'interopérabilité et l'adaptabilité des intergiciels asynchrones pour les systèmes de systèmes (thèse de J. Labéjof [80]), et
- l'informatique multi-nuages (thèse de F. Paraiso [174]).

Mes principales contributions sont :
- deux standards OMG *CORBA Scripting Language* [116] et *CORBA Components* [117],
- deux brevets avec la société Gemplus [173] et la société Thales [79],
- l'ouvrage de référence " CORBA : des concepts à la pratique " [56, 57],
- une centaine de publications (*cf.* Bibliographie),
- le modèle Services Composants Aspects (*cf.* Chapitre 2),
- la notion d'intergiciel d'intergiciels (*cf.* Chapitre 2),
- quatre logiciels d'envergure sous licence libre : CorbaScript, OpenCCM, FDF et FraSCAti.

3.2 Perspectives

En moins de dix ans, l'informatique dans les nuages, ou Cloud Computing, est devenue une réalité dans la plupart des domaines métiers. Le Cloud Computing permet de provisionner, gérer et superviser des ressources numériques en tant que services externalisés, loués à la demande, payés à l'usage et élastiques [110]. Toutefois, il n'existe pas un nuage unique et standardisé mais plutôt une multitude d'offres de nuage. Chaque offre propose ses propres modalités permettant aux utilisateurs d'allouer des ressources numériques (CPU, mémoire, réseau, stockage, serveur d'applications, base de données, etc.) à la demande puis de les administrer à distance. D'un côté, cette diversité des offres a permis de booster le marché de l'informatique en nuage. D'un autre côté, l'utilisateur de ressources en nuages fait face à quatre obstacles majeurs :

1. L'**hétérogénéité** des offres en nuage et des types de ressources en nuage. Par exemple, une ressource `Compute` chez Amazon n'a pas les mêmes caractéristiques d'une ressource `Compute` chez Google.
2. L'**interopérabilité** entre nuages. Par exemple, l'offre Amazon est accessible via une API SOAP tandis que beaucoup d'autres offres se basent sur une API REST. Toutefois, toutes ces APIs sont différentes et donc incompatibles.
3. L'**intégration** de plusieurs nuages pour construire des systèmes multi-nuages. Cette intégration est difficile due à l'hétérogénéité des offres et leur manque d'interopérabilité.
4. La **portabilité** des applications et des données sur différents nuages. Par exemple, chaque offre propose ses propres APIs pour développer les applications rendant celles-ci difficilement portables d'un nuage à un autre. La migration de données d'un nuage à un autre n'est pas facilitée car les fournisseurs n'ont aucun intérêt commercial pour cela.

Ainsi, il y a un besoin indéniable pour fournir une abstraction entre les applications et les divers nuages. Cette abstraction doit masquer l'hétérogénéité entre les offres, permettre l'interopérabilité entre nuages, faciliter la construction d'applications multi-nuages et offrir une portabilité totale sur différents nuages. Pour moi, cette abstraction est un **intergiciel pour le Cloud Computing** comme l'illustre la figure 3.1. La thèse de Fawaz Paraiso décrite dans la section 2.12 a proposé un tel intergiciel. Mon projet de recherche propose d'aller plus loin dans la quête de l'intergiciel pour le Cloud Computing.

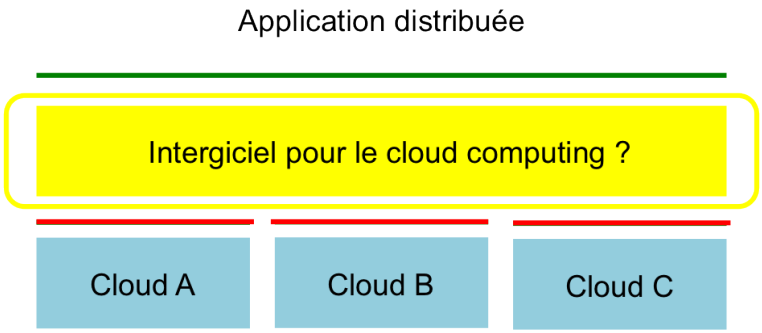

FIGURE 3.1 – Vers un intergiciel pour le Cloud Computing.

On constate un fort cloisonnement entre les ressources d'infrastructure (IaaS), de plates-formes (PaaS) et d'applications métiers (SaaS). D'un côté, chacun de ces types de ressources est géré par un gestionnaire de ressources particulier. Ces gestionnaires s'ignorent et ne savent pas collaborer. Ainsi il est extrêmement difficile de mettre en place des politiques de gestion de ressources multi-niveaux. Par exemple, gérer finement l'élasticité d'un système nécessite de gérer simultanément ces trois niveaux de ressources (IaaS, PaaS, SaaS). D'un autre côté, les applications métiers sont encore rarement organisées en ressources provisionnables à la demande. Je propose de **décloisonner les couches du Cloud Computing** et de **ressourcer les applications**.

Mon projet de recherche porte sur l'**ingénierie dirigée par les modèles pour le Cloud Computing**. Quelques projets de recherche et développement européens (REMICS, MODAClouds, SeaClouds, PaaSage, CloudML) ou américains (Eclipse Winery, StratusML) addressent l'approvisionnement et le déploiement d'applications distribuées dans les nuages grâce à une approche d'ingénierie dirigée par les modèles. Toutefois à ma connaissance, aucun projet ne propose de construire les infrastructures intergicielles en nuage via l'ingénierie dirigée par les modèles.

Mon programme de recherche est constitué des quatre axes suivants.

Modèle formel de ressources en nuage A ma connaissance, il n'existe pas de modèle formel pour modéliser tout type de ressources en nuage et raisonner sur celles-ci. Mon projet est de définir le **premier modèle formel pour raisonner sur tout type de ressources en nuages**.

Ce modèle permettra de modéliser des types de ressources. Un type de ressources identifiera les propriétés communes à une catégorie de ressources, les relations avec d'autres ressources ainsi que les actions applicables sur celles-ci, par exemple le type `Compute` avec les attributs `cpu`, `memory`, etc. et les actions `start`, `suspend`, `stop`, etc. Ce modèle permettra de typer précisément les valeurs des attributs, par exemple `Hertz` ou `Byte`. Ce modèle permettra de définir précisément les relations entre ressources comme par exemple une relation de contenance (par ex. un cluster contient des machines) ou d'association (par ex. une machine est connectée à plusieurs réseaux). Ce modèle permettra de définir le comportement des actions via par exemple des automates. Ce modèle permettra de classifier les types afin de factoriser les caractéristiques communes, par exemple les types `PhysicalMachine`, `VirtualMachine`, `DockerContainer` pourraient être des spécialisations du type abstrait `Compute`. Ce modèle permettra d'exprimer des caractéristiques transverses à plusieurs types de ressources comme

par exemple le propriétaire d'une ressource, ses utilisateurs, des accords de niveau de service (SLA), etc. Ce modèle permettra de modéliser des configurations de ressources. Une configuration sera un ensemble d'instances de ressources mises en relation et dont les attributs sont configurés.

Ce modèle sera encodé dans des langages formels afin de pouvoir appliquer des techniques formelles de validation et vérification. Dans un premier temps, j'envisage d'encoder ce modèle via le langage de spécification formelle Alloy. D'autres langages formels seront utilisés en fonction des besoins en validation et vérification.

Langage de description d'architecture pour ressources en nuage Au dessus de ce modèle formel, je propose de concevoir le **premier langage de description d'architecture dédié ressource en nuage**.

Cet ADL sera pourvu de plusieurs notations. Des notations textuelles permettront d'exprimer concrètement les types de ressources et les configurations. Des notations graphiques permettront de visualiser les types et les configurations de ressources. Pour visualiser des types, je pense qu'une représentation en 2D devrait suffire. Par contre, la visualisation de complexes configurations nécessitera surement d'investiguer des représentations en 3D voire plus.

Ces notations, ou langages spécifiques (DSL), s'appuieront sur les techniques de l'ingénierie dirigée par les modèles afin de rationaliser et faciliter leur mise en œuvre.

Catalogue de modèles de ressources en nuage L'objectif de ce troisième axe sera de constituer le **premier catalogue de modèles de ressources en nuage**.

Ce catalogue contiendra des modèles de ressources IaaS, PaaS et SaaS. Au niveau IaaS, ce catalogue contiendra les modèles des ressources fournies par toutes les offres du marché ou au minimum les offres significatives telles que celles d'Amazon, Google, VMware, OpenStack, etc. Ce catalogue couvrira aussi les ressources de type *Network as a Service*, *Machine as a Service* puis *DataCenter as a Service*. Au niveau PaaS, le focus sera mis sur un modèle de ressources pour les conteneurs Docker ainsi que des modèles pour le *Big Data as a Service* et *Linked Data as a Service*.

Ce catalogue sera mis en ligne dans les nuages avec l'objectif de devenir le catalogue de référence.

Gestionnaire de ressources élastiques dirigé par les modèles L'objectif de cet axe sera de construire le **premier gestionnaire de ressources élastiques dirigé par les modèles**.

Ce gestionnaire sera générique et capable d'héberger tout type de ressources en nuage. En fait, ce gestionnaire sera un interpréteur des modèles de ressources précédents. Ce gestionnaire s'inscrit ainsi dans la démarche Models@run.time [14]. Ce gestionnaire hébergera des connecteurs assurant le lien causal entre les modèles de ressources et les véritables ressources. Ce gestionnaire permettra accueillir des politiques de gestion de l'élasticité des ressources. Ces politiques pourront prendre en compte les différents niveaux de ressources IaaS, PaaS et SaaS.

J'ai commencé à travailler sur ces quatre perspectives [118]. Courant 2014, j'ai initié et monté le projet PIA OCCIware[1] [179]. Actuellement, je suis le responsable scientifique et technique de ce projet jusqu'à fin 2017. Ce projet a pour objectif de construire un cadre formel et outillé pour la gestion de toute ressource en nuage. Ce cadre s'appuie sur les recommandations *Open Cloud Computing Interface* (OCCI) proposées par l'*Open Grid Forum* (OGF). Actuellement, je suis en train de monter un Laboratoire Commun avec la société Scalair, fournisseur de Cloud et partenaire du projet PIA OCCIware. Nous partageons une vision commune : **les prochaines générations d'infrastructures en nuage seront pilotées par des modèles bien fondés**. Ce laboratoire commun porte ainsi sur l'ingénierie dirigée par les modèles pour l'informatique en nuage et vise à construire un gestionnaire de conteneurs Docker élastiques dirigé par des modèles bien fondés.

1. http://www.occiware.org

Bibliographie

[1] Takoua Abdellatif : *Apport des architectures à composants pour l'administration des intergiciels.* Thèse de doctorat, Institut National Polytechnique de Grenoble, septembre 2006.

[2] Mathieu Acher, Anthony Cleve, Philippe Collet, Philippe Merle, Laurence Duchien et Philippe Lahire : Reverse Engineering Architectural Feature Models. *In Proceeding of 5th European Conference of Software Architecture, ECSA 2011*, volume 6903 de *Lecture Notes in Computer Science (LNCS)*, pages 220–235. Springer, Essen, Germany, septembre 2011.

[3] Mathieu Acher, Anthony Cleve, Philippe Collet, Philippe Merle, Laurence Duchien et Philippe Lahire : Extraction and Evolution of Architectural Variability Models in Plugin-based Systems. *Software & Systems Modeling (SoSyM)*, 13(4):1367–1394, octobre 2014.

[4] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire et Philippe Merle : Feature Model Differences. *In Proceedings of 24th International Conference on Advanced Information Systems Engineering, CAiSE 2012*, volume 7328 de *Lecture Notes in Computer Science (LNCS)*, pages 629–645. Springer, Gdańsk, Poland, juin 2012.

[5] Jonathan Aldrich, Craig Chambers et David Notkin : ArchJava : connecting software architecture to implementation. *In ICSE 2002. Proceedings of the 24rd International Conference on Software Engineering, 2002*, pages 187–197. IEEE, 2002.

[6] Gustavo Alonso, Fabio Casati, Harumi Kuno et Vijay Machiraju : *Web Services.* Springer, 2004.

[7] Frederico Guilherme Alvares De Oliveira Junior : *Multi Autonomic Management for Optimizing Energy Consumption in Cloud Infrastructures.* Thèse de doctorat, Université de Nantes, avril 2013.

[8] David Basin, Manuel Clavel et Marina Egea : A Decade of Model-Driven Security. *In Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT'11)*, pages 1–10. ACM, 2011.

[9] David Basin, Jürgen Doser et Torsten Lodderstedt : *Model Driven Security.* Springer, 2005.

[10] Françoise Baude, Imen Filali, Fabrice Huet, Virginie Legrand Contes, Elton N. Mathias, Philippe Merle, Cristian Ruz, Reto Krummenacher, Elena Paslaru Bontas Simperl, Christophe Hammerling et Jean-Pierre Lorré : ESB Federation for Large-Scale SOA. *In Proceedings of the 2010 ACM Symposium on Applied Computing (SAC 2010)*, pages 2459–2466. ACM, mars 2010.

[11] Françoise Baude, Ludovic Henrio et Cristian Ruz : Programming distributed and adaptable autonomous components—the GCM/ProActive framework. *Software : Practice and Experience*, 2014.

[12] Michael Beisiegel, Henning Blohm, Dave Booz, Mike Edwards, Oisin Hurley, S Ielceanu, A Miller, A Karmarkar, A Malhotra, J Marino *et al.* : SCA Service Component Architecture - Assembly Model Specification. Version 1.0, 2007.

[13] Philip A. Bernstein : Middleware : A Model for Distributed System Services. *Communication of ACM*, 39(2):86–98, février 1996.

[14] Gordon Blair, Nelly Bencomo et Robert B France : Models@run.time. *Computer*, 42(10):22–27, octobre 2009.

[15] Marc Born, Tom Ritter et Philippe Merle : Realization of Distributed Applications using MDA and the CORBA Component Model. Tutoriel durant 4th ACM/IFIP/USENIX International Middleware Conference, Middleware'03, juin 2003.

[16] Alain Boulze, Stéphane Bagnier, Julien Forrest, Sébastien Jourdain, Mohammed El Jai, Claude Meynier, Jérôme Besnainou, Marc Dutoo, Marcel Arrufat Arias, Adrian Mos et Guillaume Vaudaux-Ruth : Usage Analysis & Demonstrators. Livrable version 2.0, ANR TLog SCOrWare Project, mars 2009.

[17] Frédéric Briclet : Canevas logiciel générique pour le déploiement dans les intergiciels pour composants. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juin 2004.

[18] Frédéric Briclet, Christophe Contreras et Philippe Merle : OpenCCM : une infrastructure à composants pour le déploiement d'applications à base de composants CORBA. *In 1ère conférence sur le Déploiement et la (Re)Configuration de Logiciels (DECOR 2004)*, pages 101 – 112, Grenoble, France, octobre 2004. Hermès Sciences.

[19] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma et Jean-Bernard Stefani : The Fractal component model and its support in Java. *Software : Practice and Experience*, 36(11-12):1257–1284, 2006.

[20] Rajkumar Buyya, Rajiv Ranjan et Rodrigo N Calheiros : Intercloud : Utility-oriented federation of cloud computing environments for scaling of application services. *In Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[21] Rajkumar Buyya, Chee Shin Yeo et Srikumar Venugopal : Market-oriented cloud computing : Vision, hype et reality for delivering it services as computing utilities. *In Proccedings of the 10th IEEE International Conference on High Performance Computing and Communications, HPCC'08*, pages 5–13. IEEE, 2008.

[22] Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Laura Ricci et Giacomo Righetti : Cloud federations in Contrail. *In Euro-Par 2011 : Parallel Processing Workshops*, pages 159–168. Springer, 2012.

[23] Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns et Jon Whittle : Software

engineering for self-adaptive systems : A research roadmap. *In Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer-Verlag, 2009.

[24] Alain Chiquet, Didier Donsez, Jacky Estublier, Colombe Hérault, Sylvain Leblanc, Sylvain Lecomte, Philippe Merle, Olivier Potonniée et T. Vessière : Architecture générale. Rapport technique 2002-1, Projet RNRT COMPiTV, 2002.

[25] Paul Clements et Linda M. Northrop : *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional Reading, 2001.

[26] Geoff Coulson, Gordon Blair, Paul Grace, Francois Taiani, Ackbar Joolia, Kevin Lee, Jo Ueyama et Thirunavukkarasu Sivaharan : A generic component model for building systems software. *ACM Transactions on Computer Systems (TOCS)*, 26(1):1, 2008.

[27] Pierre-Charles David : *Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation*. Thèse de doctorat, Université de Nantes, juillet 2005.

[28] Pierre-Charles David, Thomas Ledoux, Marc Léger et Thierry Coupaye : FPath and FScript : Language support for navigation and reliable reconfiguration of Fractal architectures. *Annals of Telecommunications*, 64(1-2):45–63, 2009.

[29] Christophe Demarey et Damien Fournier : FraSCAti, prenez le contrôle sur vos applications. *Programmez !*, 136, décembre 2010.

[30] Christophe Demarey et Damien Fournier : SOA facile avec SCA. *Programmez !*, 135, novembre 2010.

[31] Stefan Dietze, Carlos Pedrinaci, Alessio Gugliotta, Philippe Merle, Iván Martínez, Guillermo Álvaro Rey, Carlos Ruiz Moreno, Matteo Villa et Sven Abel : Service Provisioning Platform Design. Deliverable D2.1.1, FP7 IST SOA4All Integrated Project, août 2008.

[32] Stéphane Drapeau, Vincent Zurczak, Damien Fournier, Philippe Merle, Aurélie Hurault, Djamel Belaid, Samir Tata et Marc Dutoo : Conception and Development Tools Specifications. Livrable version 2.0, ANR TLog SCOrWare Project, mars 2009.

[33] Jérémy Dubus : Modélisation et génération automatique d'infrastructures de déploiement d'applications réparties. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juin 2005.

[34] Jérémy Dubus : *Une démarche orienté modèle pour le déploiement de systèmes en environnements ouverts distribués*. Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, octobre 2008.

[35] Jérémy Dubus et Philippe Merle : Applying OMG D&C Specification and ECA Rules for Autonomous Distributed Component-based Systems. *In Proceedings of the MoDELS Workshop on Models@Runtime*, volume 4354 de *Lecture Notes in Computer Science (LNCS)*, pages 242–251, Genova, Italia, octobre 2006. Springer-Verlag.

[36] Jérémy Dubus et Philippe Merle : Autonomous Deployment and Reconfiguration of Component-Based Applications in Open Distributed Environments. *In Proceedings of the 8th International OTM Symposium on Distributed Objects and Applications (DOA'06)*, volume 4277 de *Lecture Notes in Computer Science (LNCS)*, pages 26–27, Montpellier, France, novembre 2006. Springer-Verlag.

[37] Jérémy Dubus et Philippe Merle : Vers l'auto-adaptabilité des architectures logicielles dans les environnements ouverts distribués. *In Actes de la 1ère Conférence Francophone sur les Architectures Logicielles (CAL'06)*, pages 13–29, Nantes, France, septembre 2006. Hermès Sciences.

[38] Jérémy Dubus et Philippe Merle : Towards Model-Driven Validation of Autonomic Software Systems in Open Distributed Environments. *In Proceedings of the ECOOP Workshop on Model-Driven Adaptation (MADAPT)*, pages 39–48, Berlin, Germany, juillet 2007.

[39] Jérémy Dubus, Areski Flissi, Nicolas Dolet et Philippe Merle : Une démarche orientée modèle pour déployer des systèmes logiciels réparties. *Revues des Sciences et Technologies de l'Information - L'Objet*, 14(1-2):35 – 59, juillet 2008.

[40] Laurence Duchien et Lionel Seinturier : Observation of Distributed Computations : a Reflective Approach for CORBA. *International Journal of Parallel and Distributed Systems and Networks*, 4(1):17–25, 2001.

[41] Bruno Dumant, François Horn, Frédéric Dang Tran et Jean-Bernard Stefani : Jonathan : an open distributed processing environment in Java. *Distributed Systems Engineering*, 6(1):3–12, 1999.

[42] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui et Anne-Marie Kermarrec : The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, juin 2003.

[43] Alexandre Feugas : *An Agile, Reliable and Minimalist Approach to Preserve the QoS of Business-Processes Based Applications during their Evolutions.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, octobre 2014.

[44] Roy T. Fielding : *Architectural Styles and the Design of Network-based Software Architectures.* Thèse de doctorat, University of California at Irvine, 2000.

[45] Areski Flissi, Jérémy Dubus, Nicolas Dolet et Philippe Merle : Deploying on the Grid with DeployWare. *In Proceedings of the 8th International Symposium on Cluster Computing and the Grid (CCGRID'08)*, pages 177–184, Lyon, France, mai 2008. IEEE.

[46] Areski Flissi, Christophe Gransart et Philippe Merle : A Component-Based Software Infrastructure for Contextual Transportation Applications. *In The 5th International Conference on Intelligent Transportation System - Telecommunication (ITS-T 2005)*, pages 307 – 310, Brest, France, juin 2005. ENST Bretagne.

[47] Areski Flissi, Christophe Gransart et Philippe Merle : A Component-Based Software Infrastructure for Ubiquitous Computing. *In Proceeding of 4th International Symposium on Parallel and Distributed Computing (ISPDC 2005)*, pages 183–190, Lille, France, juillet 2005. IEEE.

[48] Areski Flissi, Christophe Gransart et Philippe Merle : A Service Discovery and Automatic Deployment Component-Based Software Infrastructure for Ubiquitous Computing. *In Ubiquitous Mobile Information and Collaboration Systems, CAiSE Workshop (UMICS 2005)*, pages 601 – 615, Porto, Portugal, juin 2005.

[49] Areski Flissi, Christophe Gransart et Philippe Merle : Une infrastructure à composants pour des applications ubiquitaires. *In Actes de la 2nde conférence Ubiquité et Mobilité (UbiMob 2005)*, pages 45–48, Grenoble, France, juin 2005. ACM.

[50] Areski Flissi et Philippe Merle : A Generic Deployment Framework for Grid Computing and Distributed Applications. *In Proceedings of the 2nd International OTM Symposium on Grid computing, high-performAnce and Distributed Applications (GADA'06)*, volume 4279 de *Lecture Notes in Computer Science (LNCS)*, pages 1402–1411, Montpellier, France, novembre 2006. Springer-Verlag.

[51] Damien Fournier, Philippe Merle, Gaël Blondelle, Lionel Seinturier, Nicolas Dolet, Christophe Demarey, Vivien Quema, Valerio Schiavoni, Samir Tata, Djamel Belaid et Daniel Hagimont : SCA Platform Specifications. Livrable version 2.0, ANR TLog SCOrWare Project, avril 2009.

[52] Sergio Garcia-Gomez, Miguel Jimenez-Ganan, Yehia Taher, Christof Momm, Frederic Junker, Jozsef Biro, Andreas Menychtas, Vasilios Rikopoulos et Steve Strauch : Challenges for the comprehensive management of Cloud Services in a PaaS framework. *Scalable Computing : Practice and Experience*, 13(3), 2012.

[53] Jean-Marc Geib, Chistophe Gransart, Chrystel Grenot et Philippe Merle : Introduction au langage BOX. Rapport technique ERA-150, LIFL, avril 1994.

[54] Jean-Marc Geib, Christophe Gransart, Chrystel Grenot et Philippe Merle : (Object and Thread)-Oriented Distributed Computing. *In Workshop Object-Based Parallel and Distributed Computation (OBPDC 1995)*, juin 1995.

[55] Jean-Marc Geib, Christophe Gransart, Chrystel Grenot et Philippe Merle : (Thread and Object)-Oriented Distributed Programming. *In Object-Based Parallel and Distributed Computation (OBPDC 1995)*, volume 1107 de *Lecture Notes in Computer Science (LNCS)*, pages 83 – 103. Springer, avril 1996.

[56] Jean-Marc Geib, Christophe Gransart et Philippe Merle : *CORBA : des concepts la pratique.* Collection Inter-Editions. Masson, octobre 1997.

[57] Jean-Marc Geib, Christophe Gransart et Philippe Merle : *CORBA : des concepts la pratique, seconde édition.* Collection Inter-Editions. Dunod, octobre 1999.

[58] Jean-Marc Geib et Philippe Merle : CORBA : des concepts à la pratique. *Techniques de l'Ingénieur, traité Informatique*, (H-2-758), février 2000.

[59] Christophe Gransart, Philippe Merle et Jean-Marc Geib : GoodeWatch : Supervision of CORBA Applications. *In Proceedings of ECOOP Workshop OOOS*, volume 1743 de *Lecture Notes in Computer Science (LNCS)*, page 26. Springer, juin 1999.

[60] Chrystel Grenot et Philippe Merle : Le langage BOX : étude et réalisation. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), septembre 1992.

[61] Nikolay Grozev et Rajkumar Buyya : Inter-Cloud Architectures and Application Brokering : Taxonomy and Survey. *Software : Practice and Experience*, 2012. http ://dx.doi.org/10.1002/spe.2168.

[62] Nicolas Haderer : *APISENSE® : une plate-forme répartie pour la conception, le déploiement et l'exécution de campagnes de collecte de données sur des terminaux intelligents.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, novembre 2014.

[63] Nicolas Haderer, Fawaz Paraïso, Christophe Ribeiro, Philippe Merle, Romain Rouvoy et Lionel Seinturier : *Cloud-based Software Crowdsourcing*, chapitre A

Cloud-based Infrastructure for Crowdsourcing Data from Mobile Devices. Lecture Notes in Computer Science (LNCS). Springer, A paraître en 2015. Séminaire Dagstuhl.

[64] Christophe Hamerling, Virginie Legrand, Francoise Baude, Elton Mathias, Cristian Ruz, Michael Fried, Reto Krummenacher, Philippe Merle et Nicolas Dolet : SOA4All Runtime. Deliverable D1.4.1B, FP7 IST SOA4All Integrated Project, septembre 2009.

[65] Leonard Heilig et Stefan Voss : A Scientometric Analysis of Cloud Computing Literature. *IEEE Transactions on Cloud Computing*, 2(3):266 – 278, juillet - septembre 2014.

[66] Gabriel Hermosillo : *Towards Creating Context-Aware Dynamically-Adaptable Business Processes Using Complex Event Processing*. Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, juin 2012.

[67] A. Hoffmann, T. Ritter, J. Reznik, M. Born, B. Neubauer, F. Stoinski, H. Boehme, B. Folliot, M. Vadet, U. Lang, P. Merle et C. Contreras : Specification of the Deployment and Configuration. Rapport technique D2.2/2.3, Specification of the Telecom CCM Infrastructure IST Programme. Project IST-2001-34445. 1 April 2002 to 31 March 2004, juillet 2003.

[68] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, Marco Aurélio Gerosa et Amira Ben Hamida : Service-Oriented Middleware for the Future Internet : State of the Art and Research Directions. *Journal of Internet Services and Applications*, 2(1):23–45, juillet 2011.

[69] M. Jamshidi : System-of-Systems Engineering - a Definition. *IEEE SMC - Systems, Man and Cybernetics*, octobre 2005.

[70] Eleni Kamateri, Nikolaos Loutas, Dimitris Zeginis, James Ahtes, Francesco D'Andria, Stefano Bocconi, Panagiotis Gouvas, Giannis Ledakis, Franco Ravagli et Oleksandr Lobunets : Cloud4SOA : A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability. *In Service-Oriented and Cloud Computing*, pages 64–78. Springer, 2013.

[71] Jeffrey O. Kephart et David M. Chess : The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[72] Bassem Kosayba, Raphaël Marvie, Philippe Merle et Jean-Marc Geib : Reconfiguration d'applications réparties à base de composants. Poster aux Journées Composants (JC'02) « Systèmes à composants adaptables et extensibles », octobre 2002.

[73] Bassem Kosayba, Raphaël Marvie, Philippe Merle et Jean-Marc Geib : Production of domain oriented graphic modeling environments. Poster au Workshop MDAFA 2003, juin 2003.

[74] Bassem Kosayba, Philippe Merle, Raphaël Marvie et Jean-Marc Geib : Production d'environnements graphiques à partir de méta-modèles. *In Actes de l'atelier de travail du groupe OCM du GDR ALP*, Vannes, France, février 2003.

[75] Olga Kouchnarenko et Jean-François Weber : Adapting Component-Based Systems at Runtime via Policies with Temporal Patterns. *In* José Luiz Fiadeiro, Zhiming Liu et Jinyun Xue, éditeurs : *FACS 2013, 10th International Symposium on Formal Aspects of Component Software, Revised Selected Papers*, volume 8348 de *LNCS*, pages 234–253, Nanchang, China, 2014. Springer. Revised Selected Papers.

[76] Sacha Krakowiak : Middleware Architecture with Patterns and Frameworks. `http://sardes.inrialpes.fr/~krakowia/MW-Book/`, 2007.

[77] Reto Krummenacher, Ioan Toma, Christophe Hamerling, Jean-Pierre Lorre, Francoise Baude, Virginie Legrand, Philippe Merle, Cristian Ruz, Carlos Pedrinaci, Dong Liu et Tomas Pariente Lobo : SOA4All Reference Architecture Specification. Deliverable D1.4.1A, FP7 IST SOA4All Integrated Project, mars 2009.

[78] Jonathan Labéjof, Antoine Léger, Philippe Merle, Lionel Seinturier et Hugues Vincent : R-MOM : A Component-Based Framework for Interoperable and Adaptive Asynchronous Middleware Systems. *In First International Workshop on Service and Cloud Based Data Integration (SCDI) at the 16th IEEE International EDOC Conference*, pages 204–213, Beijing, China, septembre 2012. IEEE.

[79] Jonathan Labéjof, Philippe Merle, Antoine Léger et Lionel Seinturier : Data Distribution System Based on the Exchange of Asynchronous Messages. US 20130024874 A1 & EP 2549715 A1, janvier 2013. Déposé par Thales.

[80] Jonathan Labéjof : *R-*, Réflexivité au service de l'Évolution des Systèmes de Systèmes*. Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, décembre 2012.

[81] Sylvain Leblanc : Courtage de composants répartis avec TORBA et TOSCA . Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juillet 2001.

[82] Sylvain Leblanc, Raphaël Marvie, Philippe Merle et Jean-Marc Geib : *Les intergiciels – développements récents dans CORBA, Java RMI et les agents mobiles*, chapitre TORBA : contrats de courtage pour CORBA, pages 47 – 72. Hermès Sciences, avril 2002. ISBN : 2-7462-0432-0.

[83] Sylvain Leblanc et Philippe Merle : TORBA : vers un serveur de composants de courtage. *In Actes des Journées Composants (JC'01) « Flexibilité du système au langage »*, pages 55 – 66, octobre 2001.

[84] Sylvain Leblanc, Philippe Merle et Jean-Marc Geib : Vers des composants de courtage avec TORBA. *RSTI - L'Objet*, 8(1-2/2002):185 – 201, janvier 2002. Actes de la conférence Langages Modèles Objets, LMO 2002.

[85] Sylvain Leblanc, Philippe Merle et Jean-Marc Geib : Les contrats de courtage TORBA. *L'Informatique Professionnelle*, (212):40–44, mars 2003. ISBN : 0750-1080.

[86] Matthieu Leclercq, Vivien Quéma et Jean-Bernard Stefani : DREAM : a component framework for the construction of resource-aware, reconfigurable MOMs. *In Proceedings of the 3rd workshop on Adaptive and Reflective Middleware*, ARM'04, pages 250–255, Toronto, Ontario, Canada, 2004. ACM.

[87] Virginie Legrand Contes : *Une approche à composant pour l'orchestration de services à large échelle*. Thèse de doctorat, Université de Nice - Sophia Antipolis, décembre 2011.

[88] Levi Lúcio, Qin Zhang, Phu H. Nguyen, Moussa Amrani, Jacques Klein, Hans Vangheluwe et Yves Le Traon : Chapter 3 - Advances in Model-Driven Security. *In* Atif Memon, éditeur : *Advances in Computers*, volume 93, pages 103—-152. Elsevier, 2014.

[89] Michal Malohlava, Aleš Plšek, Frédéric Loiret, Philippe Merle et Lionel Seinturier : Introducing Distribution into a RTSJ-based Component Framework. *In Proceedings of the 2nd Researcher Workshop on Real-Time Computing (JRWRTC'08)*, page 4, Rennes, France, octobre 2008.

[90] Jim Marino et Michael Rowley : *Understanding SCA (Service Component Architecture)*. Pearson Education, 2009.

[91] Raphael Marvie et Philippe Merle : Vers un modèle de composants pour cesure, le corba component model. Rapport technique RNRT 98 CESURE-3, Projet CESURE, novembre 2000.

[92] Raphaël Marvie : Structures d'accueil pour la mobilité et le cache de composants métier. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juillet 1999.

[93] Raphaël Marvie : *Séparation des préoccupations et méta-modélisation pour environnements de manipulation d'architectures logicielles à base de composants*. Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, décembre 2002.

[94] Raphaël Marvie et Philippe Merle : CODeX : proposition pour la description dynamique d'architectures à base de composants logiciels. *In Actes des Journées Composants (JC'01) « Flexibilité du système au langage »*, pages 79 – 88, octobre 2001.

[95] Raphaël Marvie et Philippe Merle : CORBA Component Model : Discussion and Use with OpenCCM. Rapport technique, LIFL, janvier 2002.

[96] Raphaël Marvie, Philippe Merle et Olivier Caron : Le modèle de composants ccm. Rapport technique 1.1, Projet RNTL Accord, avril 2002.

[97] Raphaël Marvie, Philippe Merle et Jean-Marc Geib : A Dynamic Platform for CORBA Component Based Applications . *In First International Conference on Software Engineering Applied to Networking and Parallel Distributed Computing (SNPD 2000)*, pages 352–357, Reims, France, mai 2000. IACIS. ISBN : 0-9700776-0-2.

[98] Raphaël Marvie, Philippe Merle et Jean-Marc Geib : Towards a Dynamic CORBA Component Platform . *In Proceedings of the 2nd International Symposium on Distributed Object Applications (DOA 2000)*, pages 305–314, Dallas, Texas, USA, septembre 2000. IEEE. ISBN : 0-7695-0819-7.

[99] Raphaël Marvie, Philippe Merle et Jean-Marc Geib : A Dynamic Platform for CORBA Component-Based Applications. Poster Middleware 2000, avril 2000.

[100] Raphaël Marvie, Philippe Merle et Jean-Marc Geib : Separation of Concerns in Modeling Distributed Component Based Architectures . *In Proceedings of the 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002)*, pages 144–154, Lausanne, Switzerland, septembre 2002. IEEE. ISBN : 0-7695-1742-0.

[101] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Christophe Gransart : Des objets aux composants CORBA, première étude et expérimentation avec CorbaScript. *In Proceedings of the 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA'99)*, volume 12, Paris, France, décembre 1999. papier 11 – 4.

[102] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Christophe Gransart : CCM + IDLscript = Applications Distribuées. *Numéro spécial « Evolution des plates-formes orientées objets répartis », Calculateurs Parallèles et Réseaux*, 12(1):75 – 104, 2000. ISBN : 2-7462-0169-0.

[103] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Sylvain Leblanc : TORBA : vers des contrats de courtage . *In Actes du 3eme Colloque International sur les NOuvelles TEchnologies de la REpartition, NOTERE 2000*, numéro ENST 2000 S 003, pages 3–20, Paris, France, novembre 2000. ENST.

[104] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Sylvain Leblanc : TORBA : Trading Contracts for CORBA. *In Proceedings of the 6th USENIX Conference on Object- Oriented Technologies and Systems (COOTS 2001)*, pages 1–14. USENIX, janvier 2001. ISBN : 1-880446-12-X.

[105] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Sylvain Leblanc : TORBA : vers des contrats de courtage. *Electronic Journal on Network and Distributed Processing (EJNDP)*, 1(11):1–18, mars 2001. ISBN : 1262-3261.

[106] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Sylvain Leblanc : Type-safe Trading Proxies Using TORBA. *In Proceedings of the 5th International Conference on Autonomous Distributed Systems (ISADS 2001)*, pages 303–310, Dallas, Texas, USA, mars 2001. IEEE. ISBN : 0-7695-1065-5.

[107] Raphaël Marvie, Philippe Merle, Jean-Marc Geib et Mathieu Vadet : OpenCCM : une plate-forme ouverte pour composants CORBA. *In Actes de la 2ème Conférence Française sur les Systèmes d'Exploitation, CFSE 2*, pages 1–12, avril 2001.

[108] Nenad Medvidovic et Richard N Taylor : A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[109] Rémi Mélisson, Philippe Merle, Daniel Romero, Romain Rouvoy et Lionel Seinturier : Reconfigurable Run-Time Support for Distributed Service Component Architectures. *In Proceedings of Automated Software Engineering (ASE, Tool Demonstration*, pages 171–172, Antwerp, Belgium, septembre 2010. IEEE/ACM.

[110] Peter Mell et Timothy Grance : The NIST Definition of Cloud Computing. *NIST Special Publication*, 800(145), septembre 2011.

[111] Marcilio Mendonca, Moises Branco et Donald Cowan : SPLOT : software product lines online tools. *In Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 761–762. ACM, 2009.

[112] P. Merle, C. Contreras, M. Born, J. Reznik, A. Hoffman, T. Ritter, A. Rennoch, B. Neubauer, S. Efremidis, M. Vadet, U. Lang et R. Schreiner : State of the Art : Component Models for Distributed Systems. Rapport technique D1.2, Requirement Analysis of Telecom CORBA Components IST Programme. Project IST-2001-34445. 1 April 2002 to 31 March 2004, mars 2003.

[113] Philippe Merle : How to Make Corba Objects User-Friendly with a Generic Object Oriented Dynamic Environment ? Rapport technique ERA-170, LIFL, mai 1996.

[114] Philippe Merle : *CorbaScript - CorbaWeb : propositions pour l'accès à des objets et services répartis.* Thèse de doctorat, Université des Sciences et Technologies de Lille

(USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, janvier 1997.

[115] Philippe Merle : CORBA Scripting Language Specification, v1.0. OMG TC Document formal/2001-06-05, OMG, juin 2001.

[116] Philippe Merle : CORBA Scripting Language Specification, v1.1. OMG TC Document formal/2003-02-01, OMG, février 2003.

[117] Philippe Merle et al : CORBA Components Specification, v3.0 . OMG TC Document formal/2002-06-65, OMG, juin 2002.

[118] Philippe Merle, Olivier Barais, Jean Parpaillon, Noël Plouzeau et Samir Tata : A Precise Metamodel for Open Cloud Computing Interface. *In Proceedings of 8th IEEE International Conference on Cloud Computing (CLOUD 2015)*, pages 852 – 859, New York, United States, juin 2015. IEEE.

[119] Philippe Merle et Serge Du : What we can get from a CORBA Scripting language in HEP. *In Proceedings of CHEP'98*, septembre 1998.

[120] Philippe Merle, Serge Du et Oleg Lodygensky : CORBA scripting in HEP and beyond. *In Proceedings of CHEP'00*, février 2000.

[121] Philippe Merle, Chirstophe Gransart et Jean-Marc Geib : CorbaWeb : A Navigator for CORBA Objects. *Dr. Dobb's SOURCEBOOK, numéro spécial « Distributed Objects »*, pages 7 – 11, janvier/février 1997.

[122] Philippe Merle, Chistophe Gransart, Jean-François Roos et Jean-Marc Geib : CorbaScript : A Dedicated CORBA Scripting Language. *In Proceedings of CHEP'98*, septembre 1998.

[123] Philippe Merle, Christophe Gransart et Jean-Marc Geib : Collection of articles on CorbaScript, CorbaWeb and the GOODE project. Rapport technique ERA-174, LIFL, novembre 1996.

[124] Philippe Merle, Christophe Gransart et Jean-Marc Geib : CorbaScript and CorbaWeb : A Generic Object-Oriented Dynamic Environment upon CORBA. *In Proceedings of TOOLS Europe'96*. Prentice-Hall, février 1996.

[125] Philippe Merle, Christophe Gransart et Jean-Marc Geib : CorbaWeb : A Generic Object Navigator. *Computer Networks and ISDN Systems*, 28(7-11):1269–1281, mai 1996. Proceedings of WWW5.

[126] Philippe Merle, Christophe Gransart et Jean-Marc Geib : CorbaWeb : A WWW and CORBA Worlds Integration. *In COOTS Workshop « Distributed Object Computing on the Internet »*. Usenix, juin 1996.

[127] Philippe Merle, Christophe Gransart et Jean-Marc Geib : Future Trends for a Global Object Web. *In Joint W3C/OMG Workshop*, juin 1996.

[128] Philippe Merle, Christophe Gransart et Jean-Marc Geib : How to Make CORBA Objects User-Friendly with GOODE ? *In ECOOP Workshop « Putting Distributed Objects to Work », Special Issues in Object-Oriented Programming*, pages 222 – 229. dpunkt-Verlag, juillet 1996.

[129] Philippe Merle, Christophe Gransart et Jean-Marc Geib : CorbaScript et CorbaWeb : concevoir, déployer et utiliser des services distribués. *In Actes du 1er Colloque International sur les NOuvelles TEchnologies de la REpartition (NOTERE'97)*, pages 53–73. TASC, novembre 1997.

[130] Philippe Merle, Christophe Gransart et Jean-Marc Geib : Des outils génériques pour exploiter CORBA. *In Actes de la Journée de Recherche sur la Conception de Systèmes Adaptatifs et Spécialisables (JRCSAS'97)*, avril 1997.

[131] Philippe Merle, Christophe Gransart et Jean-Marc Geib : Generic Tools : a New Way to Use CORBA. *In ECOOP Workshop CORBA*, juin 1997.

[132] Philippe Merle, Christophe Gransart et Jean-Marc Geib : Using and Implementing CORBA Objects with CorbaScript. *In Workshop OBPDC'97*, octobre 1997.

[133] Philippe Merle, Christophe Gransart et Jean-Marc Geib : *Object-Oriented Parallel and Distributed Programming*, chapitre Using and Implementing CORBA Objects with CorbaScript, pages 251 – 270. Hermès, janvier 2000.

[134] Philippe Merle, Romain Rouvoy et Lionel Seinturier : A Reflective Platform for Highly Adaptive Multi-Cloud Systems. *In Proceedings of the 10th International Workshop on Adaptive and Reflective Middleware (ARM'2011) at the 12th ACM/IFIP/USENIX International Middleware Conference*, pages 14 – 21, Lisbonne, Portugal, décembre 2011. ACM.

[135] Philippe Merle, Romain Rouvoy et Lionel Seinturier : FraSCAti : Adaptive and Reflective Middleware of Middleware. Tutoriel durant 12th ACM/IFIP/USENIX International Middleware Conference, Middleware'11, décembre 2011.

[136] Philippe Merle et Jean-Bernard Stefani : A formal specification of the Fractal component model in Alloy. Research Report 6721, INRIA, novembre 2008.

[137] Philippe Merle, Jean-Jacques Vandewalle et Eric Dufresne : Intégration d'environnements hétérogènes : World Wide Web, carte à microprocesseur et CORBA. *In Actes du congrès INFORSID'96*, pages 235–248, juin 1996.

[138] Philippe Merle, Jean-Jacques Vandewalle et Eric Dufresne : Intégration d'environnements hétérogènes : World Wide Web, carte à microprocesseur et CORBA. *Ingénierie des Systèmes d'Information*, 5(3):287–306, 1997.

[139] MODAClouds : MODACLOUDS architecture. `http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D3.2.1_MODACloudsArchitectureInitialVersion.pdf`, avril 2013.

[140] Naouel Moha, Francis Palma, Mathieu Nayrolles, Benjamin Joyen Conseil, Yann-Gaël Guéhéneuc, Benoit Baudry et Jean-Marc Jézéquel : Specification and Detection of SOA Antipatterns. *In Service-Oriented Computing, Proceedings of 10th International Conference, ICSOC 2012, Shanghai, China, November 12-15, 2012*, volume 7636 de *Lecture Notes in Computer Science (LNCS)*, pages 1–16. Springer Berlin Heidelberg, novembre 2012.

[141] Francesco Moscato, Rocco Aversa, Beniamino Di Martino, Dana Petcu, Massimiliano Rak et Salvatore Venticinque : An Ontology for the Cloud in mOSAIC. *IN CLOUD COMPUTING : METHODOLOGY, SYSTEM AND APPLICATIONS, Taylor&Francis Group, to be published*, 2011.

[142] Russel Nzekwa : *Building manageable autonomic control loops for large scale systems.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, juillet 2013.

[143] OASIS : Web Services Security : SOAP Message Security 1.1 (WS-Security 2004). OASIS Standard Specification, février 2006.

[144] OASIS : Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1. OASIS Standard, juin 2007.

[145] OASIS : Service Component Architecture WS-BPEL Client and Implementation Specification Version 1.1. OASIS Committee Draft 02 / Public Review Draft 01, mars 2009.

[146] OASIS : Web Services Dynamic Discovery (WS-Discovery) Version 1.1. OASIS Standard, juillet 2009.

[147] OASIS : WS-SecureConversation 1.4. OASIS Standard, février 2009.

[148] OASIS : Service Component Architecture Client and Implementation Model for C Specification Version 1.1. OASIS Committee Draft 06, octobre 2010.

[149] OASIS : Service Component Architecture Client and Implementation Model for C++ Specification Version 1.1. OASIS Committee Draft 06, octobre 2010.

[150] OASIS : Service Component Architecture EJB Session Bean Binding Specification Version 1.1. OASIS Committee Draft 02, février 2010.

[151] OASIS : Service Component Architecture JCA Binding Specification Version 1.1. OASIS Committee Draft 04 / Public Review 02, avril 2010.

[152] OASIS : SCA Policy Framework Version 1.1. OASIS Committee Specification Draft 05 / Public Review Draft 03, décembre 2011.

[153] OASIS : Service Component Architecture Assembly Model Specification Version 1.1. OASIS Committee Specification Draft 09 / Public Review Draft 04, septembre 2011.

[154] OASIS : Service Component Architecture POJO Component Implementation Specification Version 1.1. OASIS Committee Specification Draft 04 / Public Review Draft 04, août 2011.

[155] OASIS : Service Component Architecture SCA-J Common Annotations and APIs Specification Version 1.1. OASIS Committee Draft 06 / Public Review Draft 04, août 2011.

[156] OASIS : Service Component Architecture Service Component Architecture JMS Binding Specification Version 1.1. OASIS Committee Specification Draft 06 / Public Review Draft 04, juillet 2011.

[157] OASIS : WS-SecurityPolicy 1.3. OASIS Standard incorporating Approved Errata 01, avril 2012.

[158] OASIS : WS-Trust 1.4. OASIS Standard incorporating Approved Errata 01, avril 2012.

[159] OASIS : Service Component Architecture Web Service Binding Specification Version 1.1. OASIS Committee Specification Draft 06, juillet 2013.

[160] OMG : UML Profile for CORBA Components, mai 2003. OMG Document mars/03-05-09.

[161] OMG : Data Distribution Service for Real-time Systems, Version 1.2. OMG Available Specification formal/07-01-01, janvier 2007.

[162] OMG : Interface Definition Language, Version 3.5. OMG Available Specification formal/2014-03-01, mars 2014.

[163] Wendpanga Francis Ouedraogo, Frédérique Biennier et Philippe Merle : Contextualised security operation deployment through MDS@run.time architecture. *In ISC*

2014 - Intelligent Service Clouds Workshop at the 12th International Conference on Services Oriented Computing 2014, Paris, France, novembre 2014. Springer. A paraître.

[164] Wendpanga Francis Ouedraogo, Frédérique Biennier et Philippe Merle : Gestion contextualisée de la sécurité : implémentation MDS@Runtime avec FraSCAti. *In Actes de la 9ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information, SARSSI 2014*, page 10, Saint-Germain-Au-Mont-d'Or (Lyon), France, mai 2014.

[165] Wendpanga Francis Ouedraogo, Frédérique Biennier et Philippe Merle : Optimizing Service Protection with Model Driven Security@run.time. *In Proceedings of the 9th International IEEE Symposium on Service-Oriented System Engineering - IEEE SOSE 2015*, pages 50–58, Redwood City, United States, mars 2015. IEEE.

[166] PaaSage : PaaSage Deliverable D1.6.1. `http://www.paasage.eu/images/documents/paasage_d161_final.pdf`, août 2013.

[167] Francis Palma, Johann Dubois, Naouel Moha et Yann-Gaël Guéhéneuc : Detection of REST Patterns and Antipatterns : A Heuristics-Based Approach. *In Service-Oriented Computing, Proceedings of 12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014*, volume 8831 de *Lecture Notes in Computer Science (LNCS)*, pages 230–244. Springer Berlin Heidelberg, novembre 2014.

[168] Francis Palma, Naouel Moha et Yann-Gaël Guéhéneuc : Detection of Process Antipatterns : A BPEL Perspective. *In 17th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013*, pages 173–177. IEEE, septembre 2013.

[169] Francis Palma, Naouel Moha, Guy Tremblay et Yann-Gaël Guéhéneuc : Specification and Detection of SOA Antipatterns in Web Services. *In Software Architecture, Proceedings of 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014*, volume 8627 de *Lecture Notes in Computer Science (LNCS)*, pages 58–73. Springer International Publishing, novembre 2014.

[170] Francis Palma, Mathieu Nayrolles, Naouel Moha, Yann-Gaël Guéhéneuc, Benoit Baudry et Jean-Marc Jézéquel : SOA ANTIPATTERNS : AN APPROACH FOR THEIR SPECIFICATION AND DETECTION. *International Journal of Cooperative Information Systems*, 22(04), 2013.

[171] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar et Frank Leymann : Service-Oriented Computing : State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, novembre 2007.

[172] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar et Frank Leymann : Service-Oriented Computing : A Research Roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[173] Pierre Paradinas, Jean-Jacques Vandewalle, Christophe Muller, Philippe Merle, Christophe Gransart et Jean-Marc Geib : Adaptation of service applications to heterogeneous execution context by means of smart cards. US 6862614 B2, mars 2005. Déposé par Gemplus.

[174] Fawaz Paraiso : *soCloud : une plateforme multi-nuages distribuée pour la conception, le déploiement et l'exécution d'applications distribuées à large échelle.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, juin 2014.

[175] Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy et Lionel Seinturier : A Federated Multi-Cloud PaaS Infrastructure. *In Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, pages 392 – 399, Hawaii, United States, juin 2012. IEEE.

[176] Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle et Lionel Seinturier : A Middleware Platform to Federate Complex Event Processing. *In Proceedings of the Sixteenth IEEE International EDOC Conference, EDOC 2012*, pages 113–122, Beijing, China, septembre 2012. IEEE.

[177] Fawaz Paraiso, Philippe Merle et Lionel Seinturier : Managing Elasticity Across Multiple Cloud Providers. *In Proceedings of 1st International workshop on multi-cloud applications and federated clouds (MultiCloud'13)*, pages 53 – 60, Prague, Czech Republic, février 2013. ACM/SPEC.

[178] Fawaz Paraiso, Philippe Merle et Lionel Seinturier : soCloud : A service-oriented component-based PaaS for managing portability, provisioning, elasticity, and high availability across multiple clouds. *Computing, Special Issue on Cloud Computing*, août 2014.

[179] Jean Parpaillon, Philippe Merle, Olivier Barais, Marc Dutoo et Fawaz Paraiso : OCCIware - A Formal and Tooled Framework for Managing Everything as a Service. *In Proceedings of the Projects Showcase @ STAF'15*, volume 1400, pages 18 – 25, L'Aquila, Italy, juillet 2015. CEUR.

[180] Carlos Parra : *Towards Dynamic Software Product Lines : Unifying Design and Runtime Adaptations.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, mars 2011.

[181] Przemyslaw Pawluk, Bradley Simmons, Michael Smit, Marin Litoiu et Serge Mankovski : Introducing STRATOS : A Cloud Broker Service. *In IEEE CLOUD*, pages 891–898, 2012.

[182] Nicolas Pessemier : *Unification des approches par aspects et à composants.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, Lille, France, juin 2007.

[183] Nicolas Pessemier, Lionel Seinturier, Thierry Coupaye et Laurence Duchien : A Model for Developing Component-based and Aspect-oriented Systems. *In Proceedings of the 5th International Symposium on Software Composition (SC'06)*, volume 4089 de *Lecture Notes in Computer Science (LNCS)*, pages 259–273, Vienna, Austria, mars 2006. Springer-Verlag.

[184] Nicolas Pessemier, Lionel Seinturier et Laurence Duchien : Components ADL and AOP : Towards a Common Approach . *In Workshop ECOOP Reflection ; AOP and Meta-Data for Software Evolution (RAM-SE 2004)*, Oslo, Norway, juin 2004.

[185] Nicolas Pessemier, Lionel Seinturier, Laurence Duchien et Thierry Coupaye : A Component-Based and Aspect-Oriented Model for Software Evolution. *International Journal of Computer Applications in Technology (IJCAT)*, 31(1-2):94–105, 2008.

[186] Dana Petcu : Portability and interoperability between clouds : challenges and case study. *In Towards a Service-Based Internet*, pages 62–74. Springer, 2011.

[187] Dana Petcu : Multi-Cloud : Expectations and Current Approaches. *In Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 1–6. ACM, 2013.

[188] Dana Petcu, Ciprian Crăciun, Marian Neagul, Silviu Panica, Beniamino Di Martino, Salvatore Venticinque, Massimiliano Rak et Rocco Aversa : Architecturing a sky computing platform. *In ServiceWave 2010 Workshop Towards a Service-Based Internet*, pages 1–13. Springer, 2011.

[189] Dana Petcu, Georgiana Macariu, Silviu Panica et Ciprian Crciun : Portable Cloud applications - From Theory to Practice. *Future Generation Computer Systems*, 29(6): 1417–1430, août 2013.

[190] Aleš Plšek : *SOLEIL : An Integrated Approach for Designing and Developing Component-based Real-time Java Systems.* Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, septembre 2009.

[191] Aleš Plšek, Frédéric Loiret, Philippe Merle et Lionel Seinturier : A Component Framework for Java-Based Real-Time Embedded Systems. *In* Valérie Issarny et Rick Schantz, éditeurs : *Proceedings of the 9th ACM/IFIP/USENIX International Middleware Conference, Middleware'08*, volume 5346 de *Lecture Notes in Computer Science (LNCS)*, pages 124–143, Leuven, Belgium, décembre 2008. Springer-Verlag.

[192] Aleš Plšek, Philippe Merle et Lionel Seinturier : A Real-Time Java Component Model. *In Proceedings of the 11th International Symposium on Object/Component/Service-oriented Real-Time Distributed Computing (ISORC'08)*, pages 281–288, Orlando, Florida, USA, mai 2008. IEEE CS Press.

[193] Aleš Plšek, Lionel Seinturier et Philippe Merle : Ambient-Oriented Programming in Fractal. *In Proceedings of the 3rd Workshop on Object Technology for Ambient Intelligence and Pervasive Systems (OT4AmI) at ECOOP'07*, pages 33–38, juillet 2007.

[194] Vivien Quéma : *Vers l'exogiciel - Une approche de la construction d'infrastructures logicielles radicalement configurables.* Thèse de doctorat, Institut National Polytechnique de Grenoble-INPG, 2005.

[195] Michel Riveill et Philippe Merle : Programmation par composants. *Techniques de l'Ingénieur, traité Informatique*, (H-2-759), novembre 2000.

[196] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio Martín Llorente, Rubén Montero, Yaron Wolfsthal, Erik Elmroth et Juan Caceres : The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4–1, 2009.

[197] Daniel Romero : *Context as a Resource : A Service-Oriented Approach for Context-Awareness.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, juillet 2011.

[198] Romain Rouvoy : Canevas pour le transactionnel dans les plates-formes à composants. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juin 2003.

[199] Romain Rouvoy : *Une démarche à granularité extrêmement fine pour la construction de canevas intergiciels hautement adaptables : application aux services de transactions.* Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, décembre 2006.

[200] Romain Rouvoy et Philippe Merle : Abstraction of Transaction Demarcation in Component-Oriented Platforms. *In Proceedings of the fourth ACM/IFIP/USENIX International Middleware Conference, Middleware 2003*, volume 2672 de *Lecture Notes in Computer Science (LNCS)*, pages 305–323, Rio de Janeiro, Brazil, juin 2003. Springer-Verlag. ISBN : 3-540-40317-5.

[201] Romain Rouvoy et Philippe Merle : GoTM - Vers un canevas transactionnel à base de composants. *RSTI - L'Objet*, 10(2-3/2004):131–146, mars 2004.

[202] Romain Rouvoy et Philippe Merle : Leveraging Component-Oriented Programming with Attribute-Oriented Programming. *In Proceedings of the 11th International ECOOP Workshop on Component-Oriented Programming (WCOP'06)*, volume 2006-11 de *Technical Report*, pages 10–18, Nantes, France, juillet 2006. Karlsruhe University.

[203] Romain Rouvoy et Philippe Merle : Using Microcomponents and Design Patterns to Build Evolutionary Transaction Services. *In Proceedings of the International ERCIM Workshop on Software Evolution (EVOL'06)*, pages 165–179, Lille, France, avril 2006.

[204] Romain Rouvoy et Philippe Merle : Description et de vérification de motifs d'architecture avec Fractal ADL. *In Proceedings of the French Conference on Langages et Modèles à Objets (LMO'07)*, pages 49–64, Toulouse, France, mars 2007. Hermés Science Publications.

[205] Romain Rouvoy et Philippe Merle : Using Microcomponents and Design Patterns to Build Evolutionary Transaction Services. *Electronic Notes in Theoretical Computer Science (ENCTS)*, 166:111–125, janvier 2007.

[206] Romain Rouvoy et Philippe Merle : Leveraging Component-Based Software Engineering with Fraclet. *Annals of Telecommunications , Special Issue on Software Components - The Fractal Initiative*, 64(1-2):65–79, janvier/février 2009.

[207] Romain Rouvoy et Philippe Merle : Rapid Prototyping of Domain-Specific Architecture Languages. *In* Magnus Larsson and Nenad Medvidovic, éditeur : *International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE'12)*, Proceedings of the 15th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE'12), pages 13–22, Bertinoro, Italy, juin 2012. ACM.

[208] Romain Rouvoy, Nicolas Pessemier, Renaud Pawlak et Philippe Merle : Apports de la Programmation par Attributs au Modèle de Composants Fractal. *In Actes des 5èmes Journées Composants (JC'06)*, pages 11–24, Perpignan, France, octobre 2006.

[209] Romain Rouvoy, Nicolas Pessemier, Renaud Pawlak et Philippe Merle : Using Attribute-Oriented Programming to Leverage Fractal-Based Developments. *In Proceedings of the 5th International ECOOP Workshop on Fractal Component Model (Fractal'06)*, pages 1 – 8, Nantes, France, juillet 2006.

[210] Romain Rouvoy, Patricia Serrano-Alvarado et Philippe Merle : A Component-Based Approach to Compose Transaction Standards. *In Proceedings of the 5th International ETAPS Symposium on Software Composition (SC'06)*, volume 4089 de *Lecture Notes in Computer Science (LNCS)*, pages 114–130, Vienna, Austria, mars 2006. Springer-Verlag.

[211] Romain Rouvoy, Patricia Serrano-Alvarado et Philippe Merle : Towards Context-Aware Transaction Services. *In Proceedings of the 6th International Conference on Distributed Applications and Interoperable Systems (DAIS'06)*, volume 4025

de *Lecture Notes in Computer Science (LNCS)*, pages 272–288, Bologna, Italy, juin 2006. Springer-Verlag.

[212] Pierre Yves Schobbens, Patrick Heymans et Jean-Christophe Trigaux : Feature Diagrams : A Survey and a Formal Semantics. *In Proceedings of 14th IEEE International Requirements Engineering Conference*, pages 139–148. IEEE, 2006.

[213] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni et Jean-Bernard Stefani : Reconfigurable SCA Applications with the FraSCAti Platform. *In Proceedings of the 6th IEEE International Conference on Service Computing (SCC'09)*, pages 268–275. IEEE, septembre 2009.

[214] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni et Jean-Bernard Stefani : A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. *Software : Practice and Experience (SPE)*, 42(5):559–583, mai 2012.

[215] Patricia Serrano-Alvarado, Romain Rouvoy et Philippe Merle : Self-Adaptive Component-Based Transaction Commit Management. *In Proceedings of the 4th International Middleware Workshop on Adaptive and Reflective Middleware (ARM'05)*, volume 116 de *ACM International Conference Proceeding Series*, pages 1–6, Grenoble, France, novembre 2005. ACM Press.

[216] A. Singhai, A. Sane et R.H Campbell : Quarterware for middleware. *In Proceedings of 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 192 – 201, 1998.

[217] Brian Cantwell Smith : *Procedural reflection in programming languages.* Thèse de doctorat, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science., 1982.

[218] Jean-Bernard Stefani : Open distributed processing : an architectural basis for information networks. *Computer Communications*, 18(11):849–862, 1995. Beyond Intelligent Systems.

[219] Dave Steinberg, Frank Budinsky, Ed Merks et Marcelo Paternostro : *EMF : Eclipse Modeling Framework.* Pearson Education, 2008.

[220] Miruna Stoicescu : *Conception et implémentation de systèmes résilients par une approche à composants.* Thèse de doctorat, Université de Toulouse, décembre 2013.

[221] Gabriel Tamura : *QoS-CARE : A Reliable System for Preserving QoS Contracts through Dynamic Reconfiguration.* Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I ; Universidad de Los Andes, mai 2012.

[222] Alban Tiberghien : Description et déploiement de grandes architectures à composants. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juin 2008.

[223] Alban Tiberghien, Philippe Merle et Lionel Seinturier : Specifying Self-configurable Component-Based Systems with FracToy. *In Proceedings of ASM, Alloy, B and Z, ABZ 2010*, volume 5977 de *Lecture Notes in Computer Science (LNCS)*, pages 91–104, Orford, Canada, février 2010. Springer.

[224] Mathieu Vadet : Les containers ouverts dans les plates-formes à composants. Mémoire de D.E.A., Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, juillet 2001.

[225] Mathieu VADET : *Un Modèle de Services Logiciels pour la Spécialisation des Intergiciels à Composants*. Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), Laboratoire d'Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, France, décembre 2004.

[226] Mathieu VADET et Philippe MERLE : Les conteneurs ouverts dans les plates-formes à composants. *In Actes des Journées Composants (JC'01) « Flexibilité du système au langage »*,, pages 33 – 46, octobre 2001.

[227] Mathieu VADET et Philippe MERLE : Adaptation des connecteurs dans le CCM. *In Actes des Journées Composants (JC'02) « Systèmes à composants adaptables et extensibles »*. ACM SIGOPS France (ASF), octobre 2002.

[228] Thomas VERGNAUD, Jérôme HUGUES, Laurent PAUTET et Fabrice KORDON : PolyORB : a schizophrenic middleware to build versatile reliable distributed applications. *In Reliable Software Technologies-Ada-Europe 2004*, pages 106–119. Springer, 2004.

[229] W3C : Web Services Addressing 1.0 - Core. W3C Recommendation 9 May 2006, mai 2006.

[230] W3C : Web Services Addressing 1.0 - SOAP Binding. W3C Recommendation 9 May 2006, mai 2006.

[231] W3C : SOAP Version 1.2 Part 0 : Primer (Second Edition). W3C Recommendation, avril 2007.

[232] W3C : SOAP Version 1.2 Part 1 : Messaging Framework (Second Edition). W3C Recommendation, avril 2007.

[233] W3C : SOAP Version 1.2 Part 2 : Adjuncts (Second Edition). W3C Recommendation, avril 2007.

[234] W3C : Web Services Addressing 1.0 - Metadata. W3C Recommendation 4 September 2007, septembre 2007.

[235] W3C : Web Services Description Language (WSDL) Version 2.0 Part 0 : Primer. W3C Recommendation 26 June 2007, juin 2007.

[236] W3C : Web Services Description Language (WSDL) Version 2.0 Part 1 : Core Language. W3C Recommendation 26 June 2007, juin 2007.

[237] W3C : Web Services Description Language (WSDL) Version 2.0 Part 2 : Adjuncts. W3C Recommendation 26 June 2007, juin 2007.

[238] W3C : Web Services Policy 1.5 - Framework. W3C Recommendation 4 September 2007, septembre 2007.

[239] W3C : Web Application Description Language. W3C Member Submission, août 2009.

[240] Guillaume WAIGNIER : *Canevas de développement agile pour l'évolution fiable de systèmes logiciels à composants et orientés services*. Thèse de doctorat, Université des Sciences et Technologie de Lille - Lille I, janvier 2010.

[241] WS-I : Basic Profile Version 1.2. Final Material 2010-11-09, novembre 2010.

[242] Sami YANGUI, Iain-James MARSHALL, Jean-Pierre LAISNE et Samir TATA : CompatibleOne : The Open Source Cloud Broker. *Journal of Grid Computing*, pages 1–17, 2013.