

EnTiMid : Un modèle de composants pour intégrer des objets communicants dans des applications à base de services

Grégory Nain

▶ **To cite this version:**

Grégory Nain. EnTiMid : Un modèle de composants pour intégrer des objets communicants dans des applications à base de services. Génie logiciel [cs.SE]. Université Rennes 1, 2011. Français. NNT : . tel-00646664v2

HAL Id: tel-00646664

https://theses.hal.science/tel-00646664v2

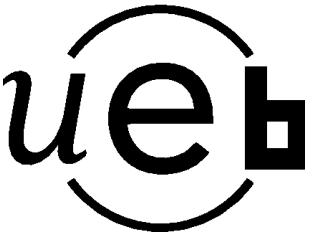
Submitted on 31 Jan 2012

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.

N° d'ordre : 4406 ANNÉE 2012





THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Grégory NAIN

préparée à l'unité de recherche
INRIA - Centre Rennes Bretagne Atlantique
Institut National de Recherche en Informatique et Automatique

---

EnTiMid : Un modèle de composants pour intégrer des objets communicants dans des applications à base de services

Thèse soutenue à Rennes
le 5 Décembre 2011

devant le jury composé de :

Daniel THOUROUDE
Professeur, IETR / Université de Rennes 1 / *Président*

Elisabetta DI NITTO
Professeur, Politecnico di Milano / *Rapporteuse*

Didier DONSEZ
Professeur, Université Joseph Fourier - Grenoble 1 / *Rapporteur*

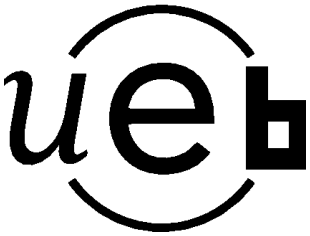
Romain ROUVOY
Maître de conférences, Université de Lille 1 / *Examinateur*

Jean-Marc JÉZÉQUEL
Professeur, Université de Rennes 1 / *Directeur de thèse*

Olivier BARAIS
Maître de conférences, Université de Rennes 1 / *Co-directeur de thèse*

N° d'ordre : 4406 ANNÉE 2012





THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Grégory NAIN

préparée à l'unité de recherche
INRIA - Centre Rennes Bretagne Atlantique
Institut National de Recherche en Informatique et Automatique

---

EnTiMid: A flexible component model to integrate smart devices in service-based applications for Ambient Assisted Living

Thèse soutenue à Rennes
le 5 Décembre 2011

devant le jury composé de :

Daniel THOUROUDE
Professeur, IETR / Université de Rennes 1 / *Président*

Elisabetta DI NITTO
Professeur, Politecnico di Milano / *Rapporteuse*

Didier DONSEZ
Professeur, Université Joseph Fourier - Grenoble 1 / *Rapporteur*

Romain ROUVOY
Maître de conférences, Université de Lille 1 / *Examinateur*

Jean-Marc JÉZÉQUEL
Professeur, Université de Rennes 1 / *Directeur de thèse*

Olivier BARAIS
Maître de conférences, Université de Rennes 1 / *Co-directeur de thèse*

Remerciements

Je voudrais commencer par remercier messieurs Jean-Marc JEZEQUEL et Olivier BARAIS pour leur soutient et leur confiance au cours de ces années passées dans l'équipe TRISKELL. Je remercie aussi l'ensemble des membres de l'équipe TRISKELL que j'ai pu croiser entre Mars 2008 et Décembre 2011, qui m'ont permis de travailler dans une équipe dynamique et chaleureuse, et ont rendu célèbre les "Barbeuk chez Greg".

Un grand merci à ma famille qui a cru en mes capacités et en ma réussite, même dans mes moments de doutes.
Merci à mes amis qui n'ont pas non plus manqué à leur rôle essentiel de soutient au cours de la thèse.

Une pensée particulière pour "*Jules*" qui a rendu la période de rédaction plus facile.

Ce n'est pas parce que c'est de la science que ça doit être chiant. G.N.
Science is fun.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Contents

1 Résumé de thèse 1
- 1.1 Rappel du contexte . . . 1
- 1.2 Résumé des exigences . . . 2
- 1.3 Étude des approches existantes . . . 4
- 1.4 Vue d'ensemble de la contribution . . . 4
- 1.5 Adéquation de la contribution . . . 6
- 1.6 Intégration à l'état de l'art . . . 7
- 1.7 Bénéfices immédiats . . . 7
  - 1.7.1 Simplification du développement de composants . . . 7
  - 1.7.2 Création d'applications par assemblage . . . 8
  - 1.7.3 Viabilité et précision . . . 8
  - 1.7.4 Intégration simplifiée d'objets et de services . . . 8
- 1.8 Limitations identifiées . . . 9
  - 1.8.1 Description structurelle . . . 9
  - 1.8.2 Paramètres de ports . . . 9
  - 1.8.3 Des vérifications basiques . . . 9
  - 1.8.4 Gestion de la variabilité . . . 10
  - 1.8.5 Absence de test sur des plate-formes embarquées . . . 10
- 1.9 Contribution au réseau d'excellence Européen S-Cube . . . 10
  - 1.9.1 Le réseau d'excellence S-Cube . . . 10
  - 1.9.2 Contribution . . . 11

I Context, Requirements and State-of-the-Art Review 13

2 Introduction 17
- 2.1 Ambient Assisted Living . . . 17
  - 2.1.1 The origins . . . 17
  - 2.1.2 The concept . . . 18
- 2.2 Home Automation . . . 19
  - 2.2.1 Application domains . . . 19
  - 2.2.2 Technologies . . . 20
- 2.3 Identification of requirements . . . 21
- 2.4 Scope of this work . . . 24

2.5 Contribution of this thesis 25

3 State-of-the-Art Review 27
3.1 Background on AAL and Home Automation 27
3.1.1 Projects in AAL 27
3.1.2 European research 29
3.1.3 Home automation in projects 29
3.1.4 Home Automation details 30
3.1.4.1 Communication Media 30
3.1.4.2 Transport Protocols 31
3.1.4.3 Application Protocols 34
3.2 General purpose approaches 37
3.2.1 (Web)Service-Oriented Architectures 37
3.2.1.1 Internet Of * and the Cloud 37
3.2.1.2 Architectural principles 39
3.2.1.3 OSGi 41
3.2.1.4 Enterprise Service Bus 42
3.2.2 Component models 43
3.2.2.1 Description 43
3.2.2.2 Darwin 43
3.2.2.3 Koala 44
3.2.2.4 Fractal 45
3.2.3 Component Models for SOA 46
3.2.3.1 Description 46
3.2.3.2 SCA 46
3.2.3.3 FraSCAti 47
3.2.3.4 iPOJO 48
3.3 Domain-specific approaches 49
3.3.1 Description 49
3.3.2 Projects 49
3.3.2.1 uMiddle 49
3.3.2.2 SOPRANO 50
3.3.2.3 Gaïa Framework 50
3.3.2.4 DiaSuite 51
3.3.2.5 Habitation 52
3.3.2.6 Wired Application Description Language 52
3.3.2.7 PervML 53
3.3.2.8 AutoHome 53
3.3.2.9 WComp 54
3.3.2.10 Niagara 55

4 Synthesis 57
4.1 Good properties identified 57
4.2 Points of contribution 59

II Thesis and Achievements 61

5 Contribution 65
5.1 Global ideas . . . 65
5.1.1 Being inspired by electronics . . . 65
5.1.2 Making it possible . . . 65
5.1.3 Keeping end-users in mind . . . 66
5.2 Overview of the contribution . . . 66

6 Details on strata 69
6.1 Device Interoperability . . . 69
6.1.1 Use of drivers . . . 70
6.1.2 Functional interfaces . . . 70
6.1.3 Event-based approach . . . 71
6.1.4 Example . . . 71
6.1.5 Threat to validity . . . 73
6.1.6 Summary . . . 73
6.2 Component Model . . . 75
6.2.1 Making software components closer to electronic components . . . 75
6.2.2 Meta-Model description . . . 76
6.2.3 Concrete example . . . 79
6.2.4 Implementation and Model Relationship . . . 80
6.2.5 Implementation independence . . . 84
6.2.6 Link with the interoperability layer . . . 85
6.2.7 Main advantage of this component model . . . 86
6.2.8 Summary . . . 86
6.3 Model@Runtime and Reasoning Engine . . . 86
6.3.1 Check to validate . . . 87
6.3.2 The Model@Runtime engine work . . . 91
6.4 Service-Oriented Runtime Architecture . . . 94
6.5 Wrappers . . . 96
6.6 Summary . . . 97

7 Outcomes 99
7.1 Implementation . . . 99
7.2 Impact on the development process . . . 100
7.2.1 Component development . . . 100
7.2.2 Application design . . . 101
7.3 Metrics . . . 101
7.4 Classification . . . 102
7.4.1 Lifecycle . . . 103
7.4.2 Constructs . . . 103
7.4.3 Extra-Functional Properties . . . 104
7.4.4 Domains . . . 104

III Validation 105

8 Validation in the context of an AAL project 109
8.1 Context of the study: the IDA project . . . 109
8.2 Use case and issues to address . . . 110
8.3 Experimental setup . . . 112
8.3.1 Delta Dore equipment . . . 113
8.3.2 KNX equipment . . . 113
8.3.3 Other equipment . . . 114
8.4 Interoperability issue . . . 114
8.4.1 Test Environment . . . 114
8.4.2 Resolution Protocol . . . 114
8.4.3 Results . . . 116
8.5 Evolution issue . . . 116
8.5.1 Test Environment . . . 116
8.5.2 Resolution Protocol . . . 116
8.5.3 Results . . . 117
8.6 Adaptation issue . . . 117
8.6.1 Test Environment . . . 118
8.6.2 Resolution Protocol . . . 118
8.6.3 Results . . . 119
8.7 Openness issue . . . 120
8.7.1 UPnP export . . . 121
8.7.1.1 Test Environment . . . 121
8.7.1.2 Resolution Protocol . . . 121
8.7.1.3 Results . . . 123
8.7.2 DPWS export . . . 123
8.7.2.1 Test Environment . . . 123
8.7.2.2 Resolution Protocol . . . 124
8.7.2.3 Results . . . 124
8.8 Threats to validity . . . 125
8.8.1 Internal threats . . . 125
8.8.1.1 Variability management . . . 125
8.8.1.2 Scalability . . . 125
8.8.1.3 Safety and Security . . . 125
8.8.2 External threats . . . 125
8.8.2.1 Validity of the scenario, real deployment . . . 125
8.8.2.2 Communications with smart devices . . . 126
8.9 Conclusion . . . 126

IV Conclusion and Perspectives 127

9 Conclusion 131

9.1 Reminder of Context . . . 131
9.2 Summary of requirements . . . 132
9.3 Survey of existing approaches . . . 133
9.4 Outline of the contribution . . . 134
9.5 Adequateness of the contribution . . . 135
9.6 Conservativeness . . . 135
9.7 Immediate benefits . . . 137
9.7.1 Development of components made easier . . . 137
9.7.2 Simple creation of applications . . . 137
9.7.3 Sustainability and precision . . . 137
9.7.4 Seamless integration of IoT and IoS . . . 137
9.8 Limitations identified . . . 138
9.8.1 Behavioral description . . . 138
9.8.2 Port parameters . . . 138
9.8.3 Too weak checkers . . . 139
9.8.4 Variability management . . . 139
9.8.5 Improvements for embedded platforms . . . 139
9.9 Contribution to the S-Cube NoE . . . 139
9.9.1 The S-Cube Network of Excellence . . . 139
9.9.2 Contribution . . . 140

10 Perspectives 141
10.1 In research . . . 141
10.1.1 IDA, second phase . . . 141
10.1.2 End User Programming . . . 141
10.1.2.1 Which description language ? . . . 141
10.1.2.2 Fuzzy Logic and Learning Algorithms . . . 142
10.1.3 Distribution and Pervasiveness . . . 142
10.1.4 Architecture Synthesis . . . 143
10.1.4.1 Dynamic Software Product Lines for the management of variability . . . 143
10.1.4.2 How can the behavior be descibed? . . . 144
10.1.5 Kevoree . . . 144
10.1.6 Open Control/Command Operating System . . . 145
10.2 In industry . . . 145
10.2.1 Public events . . . 146
10.2.2 Industrial perspectives . . . 147

V Appendix 149

A ITI Project 151
A.1 Presentation and Goals of the project . . . 152
A.1.1 Phase 1 . . . 152

A.1.2 Phase 2 . . . 152
A.1.3 Phase 3 . . . 154
A.2 Environment of tests . . . 154
A.2.1 Population under test . . . 154
A.2.2 Equipments . . . 154
A.3 Protocol of test . . . 154
A.4 Threats to validity . . . 155
A.5 Results and conclusion . . . 155

Acronyms 160

Bibliography 161

Table of figures 170

Summary 174

Chapitre 1

Résumé de thèse

Ce chapitre offre un résumé de la thèse défendue dans ce document. Pour ce faire, une introduction au contexte de ce travail, permettra de mettre en évidence les besoins que la contribution de cette thèse s'est attachée à combler. Puis, la contribution est décrite et discutée en termes d'adéquation aux besoins. Enfin, une mise en évidence des bénéfices immédiats et de quelques limitations identifiées termine ce chapitre.

1.1 Rappel du contexte

Le vieillissement de la population Européenne a incité la communauté à rechercher des solutions pour accompagner ce changement. Dans ce contexte, plusieurs problèmes doivent être considérés de front. D'abord, le domaine de la santé souffre d'une pénurie de main d'œuvre, qui pourrait résulter en une dégradation générale de la qualité des soins. Par ailleurs, les places en centres hospitaliers ou maisons de retraites sont limitées, et pourraient arriver à saturation dans les années à venir. De plus, les séjours hospitaliers coûtent chère, indépendamment de la pathologie, et les aides financières à ce niveau tendent à être limitées.
Plusieurs projets ont déjà été lancés pour de tenter de répondre à ces problématiques. Le programme collaboratif Européen *Ambient Assisted Living*(AAL) a été créé pour favoriser et financer des projets, ce qui met en évidence l'intérêt de l'Europe pour les avancées dans ce domaine. Le projet *Innovation Domicile Autonomie*(IDA), initié par la métropole Rennaise, s'inscrit parfaitement dans ce cadre. Il vise une évaluation de la pertinence d'utiliser des Technologies de l'Information et de la Communication(TIC) pour aider les personnes âgées.
Après un état des lieux précis, en termes de besoin des personnes âgées, le projet s'est efforcé de mesurer la pertinence et l'adéquation de différentes technologies industrielles, afin d'assister les personnes dans leurs domiciles. Entre autres, les technologies de la domotique ont été évaluées afin de faire ressortir leurs potentiels apports dans l'accompagnement à domicile. Rapidement, les études ont montré qu'une unique solution ne peut pas être mise en place dans tous les cas. Chaque personne à des exigences et des besoins différents, qui imposent que les solutions soient adaptées à chacun. Aussi, les

industriels admettent ici leurs limites, où là fabrication de produits personnalisés pour chaque utilisateur est trop coûteuse.

Dans ce domaine, les solutions techniques imaginées ont besoin de systèmes logiciels pour combler le vide, entre les produits finis issus du marché de la domotique, et les solutions personnalisées. Pour remplir leur mission, ces systèmes logiciels doivent satisfaire à plusieurs exigences

1.2 Résumé des exigences

L'interopérabilité est la première exigence que les systèmes logiciels ont à prendre en compte. En effet, les solutions proposées pour améliorer et favoriser le confort des personnes âgées dans leurs logements, peuvent être composées de multiple produits provenant de divers fabricants. Chaque produit prenant part à la solution, s'atèle à répondre à un des besoins de la personne de la façon la plus précise possible, rapprochant ainsi la proposition globale de la proposition idéale. Cependant, les éléments de la solution doivent aussi être capables de communiquer les uns avec les autres, afin de rendre un service global. Mais la diversité des constructeurs fait de l'interopérabilité des produits un problème de taille.
La définition d'une interface de communication, universelle à tous les composants du système, pourrait résoudre ce problème, mais requière une réingénierie de l'ensemble des produits pour les rendre compatibles. En conséquence, aucun produit actuellement sur le marché ne pourrait être utilisé. Tant que les solutions seront non limitées en termes de produits, cette proposition ne sera pas applicable, et l'interopérabilité restera une préoccupation majeure.

L'adaptation et l'évolution sont aussi des facultés essentielles pour ces systèmes. Les systèmes logiciels travaillant à partir d'objets ou d'équipements, liés à des actions du quotidien, doivent prendre en considération l'environnement dans lequel ils s'exécutent. Ils doivent être capables de s'adapter aux changements pendant leur exécution, afin de maintenir des niveaux de services et de fonctionnalités suffisant. Évidemment, ces adaptations ne doivent pas nécessiter de redémarrage du système, qui rendrait indisponible les fonctions d'alerte ou d'alarme par exemple.
Par ailleurs, les besoins, les usages, les protocoles et les technologies évoluent. Des fonctionnalités précédemment installées peuvent devenir inutiles, alors que d'autres deviennent nécessaires. La sécurité, la fiabilité du système, les protocoles peuvent être améliorés et mis à disposition dans de nouvelles versions, qui seront à prendre en compte sans avoir à ré-implémenter le système entier. Enfin, les systèmes logiciels doivent, dans ce domaine, être capable de supporter des évolutions futures non prévues au moment du déploiement, comme l'installation de nouveaux services ou fonctionnalités.

Les **contrôles distants** peuvent être nécessaires pour des questions de maintenance, de vérification de l'état de la maison par des professionnels accrédités, ou réaliser des

actions à distance pour assister la personne sur des problèmes ponctuels.

L'ouverture doit permettre à des applications tierces, d'accéder aux fonctionnalités ou aux produits disponibles dans le système. En effet, la passerelle de communication avec les réseaux KNX, par exemple, n'admet qu'une seule connexion à la fois. Il est donc nécessaire de rendre les produits et fonctionnalités disponibles à d'autres applications, afin de ne pas verrouiller les accès. Par ailleurs, cette ouverture au monde extérieur permet à des applications tierces de se connecter aux produits, et rendre des services à valeur ajoutée, sans avoir besoin de connaître l'organisation interne du système de gestion d'accès aux périphériques. L'ouverture permet donc d'enrichir l'application par des contributions externes apportant des fonctionnalités "intelligentes" supplémentaires.

La distribution est une préoccupation que l'on retrouve au cœur de tout système domotique largement déployé. La dispersion des équipements dans l'habitat mais aussi la dispersion des déploiements à l'échelle d'une ville rendent nécessaire la prise en compte des challenges des grands systèmes distribués.

La gestion de la variabilité est une préoccupation liée au fort besoin de personnalisation des solutions. En effet, l'ensemble des options déployées sur les systèmes à l'échelle d'une ville, peut devenir complexe. De plus, les évolutions ne sont pas uniformément déployées à l'échelle d'une ville, menant alors à une grande diversité de combinaisons de versions de protocoles, et d'assemblages de produits.
Des outils doivent donc être mis à disposition par ces systèmes, pour assister les ingénieurs et techniciens dans la fabrication et la maintenance de solutions. Des outils d'aide à la décision basés sur une liste d'exigences et de produits disponibles pourraient, par exemple, être d'une aide précieuse.

La sûreté est un élément important dans les systèmes domotiques. Ces systèmes ont la responsabilité de gérer une partie de la maison, afin d'améliorer la vie de ses occupants. Un niveau de service minimum doit être garanti afin que les personnes aidées par ces systèmes ne se retrouvent pas bloquées en cas d'urgence par exemple.

La sécurité est une autre préoccupation que l'on retrouve dans la construction de solutions domotiques. En effet, les accès au système doivent être sécurisés afin d'éviter qu'il soit contrôlé par des personnes non autorisées. Le compromis difficile est toujours de rendre cette sécurité transparente pour les personnels habilités.

L'acceptabilité et l'accessibilité de ces systèmes par tous les utilisateurs sont des facultés à prendre en compte, particulièrement dans le cadre d'une aide à domicile pour des personnes âgées ou dépendantes. Ces systèmes logiciels sont utilisés à la fois par les aidants à domicile, et par les personnes âgées, et pour ni l'une ni l'autre le système ne doit être perçu comme une contrainte supplémentaire dans leur métier ou leur vie. Enfin, les personnes âgées doivent pouvoir rester maître de leur environnement, et donc, garder la main sur le système et ses actions.

1.3 Étude des approches existantes

Parmi l'ensemble des exigences listées dans la section 1.2, l'étude des approches existantes ici résumée, s'est concentrée sur les aspects d'interopérabilité, d'adaptation, d'évolution, d'ouverture et de gestion de la variabilité.

De nombreuses approches s'intéressant à la résolution de problèmes d'interopérabilité, d'adaptation ou de contrôle distant pour différentes applications, sont détaillées dans la littérature scientifique.
D'une manière générale, les approches s'appuyant sur des architectures à base de services [All11, Cha04] semblent être adaptées pour résoudre des problèmes d'adaptation dynamique et d'interopérabilité, mais manquent clairement de moyens de description de l'architecture du logiciel pendant son exécution. Ils apportent aussi des mécanismes essentiels pour faire face aux apparitions et disparitions de produits, de par le fait qu'un service peut être démarré ou arrêté à tout instant.
Les architectures s'appuyant sur le paradigme de composants logiciels [GMK02, RvdLKM00, BCL$^+$06], offrent pour leur part un niveau d'abstraction interessant pour représenter les produits domotiques. Cependant, les ports de communication des composants sont souvent définis par l'intermédiaire d'interfaces de programmation qui rendent impossible certaines connections non prévues à l'avance, sans ajout de connecteurs *ad-Hoc*.
Toutefois, le mélange de l'approche à composant, et l'approche des architectures à base de services, identifié comme des composants pour les architectures orientées services [sca, MRRS10, EHL07], parait être l'approche qui réponde le mieux aux besoins identifiés ; là où les architectures à base de services offrent des solutions pour l'adaptation dynamique, les architectures à composants proposent un niveau d'abstraction interessant pour les produits domotiques, et des outils de description de l'architecture.
De façon orthogonale à toutes ces approches, les méthodes et techniques issues de l'ingénierie dirigée par les modèles, offrent des outils de manipulation et de gestion des éléments de systèmes, tant au cours du design que de l'exécution. Ils semblent apporter une réponse appropriée à la gestion de la variabilité, à la description des systèmes.

Toutes les approches et outils considérés dans cette étude sont reportés dans le tableau 1.1 présenté en section 1.5, qui présente une synthèse des points forts de chaque outils ou approche, par rapport aux exigences identifiées.

1.4 Vue d'ensemble de la contribution

Inspirée par les réalisations dans le domaine de l'électronique, cette thèse contribue à améliorer la flexibilité des systèmes logiciels tout en maintenant un haut niveau de fiabilité. Les contributions se font à trois niveaux.

(1) Un nouveau modèle de composants qui améliore la flexibilité des applications et permet la connexion de composants hétérogènes

(2) Des outils issus de l'ingénierie logicielle dirigée par les modèles (IDM), pour créer, éditer, simuler et valider la structure et le comportement des assemblages de composant avant leurs (re-)déploiements

(3) Un environnement d'exécution construit sur les bases d'une architecture logicielle orientée service, supportant les propriétés d'adaptation, d'évolution et d'ouverture requises par le nouveau modèle de composants.

L'implémentation de cette contribution, appelée EnTiMid, se compose de plusieurs éléments. Présentés sous forme de couches, ces éléments s'affairent chacun à résoudre une préoccupation particulière dans le problème global.

La couche d'**Interopérabilité des produits** est responsable de la communication avec les produits physiques, et avec leurs représentants dans le modèle de composants. Ces communications sont assurées par un ensemble de pilotes, chargés de la communication avec les produits, et de communications basées sur des messages asynchrones, qui permettent une connexion de composants réputés incompatibles.

La couche **Modèle de composants** apporte les abstractions nécessaires à la représentation et à la manipulation des produits par un architecte logiciel. Elle offre un moyen de décrire de façon unifiée les actions possibles sur les composants (et donc sur les produits), et les informations disponibles, à travers le paradigme de ports. Dans ce modèle, les ports des composants peuvent être de deux sortes : synchrones (ports de service) ou asynchrones (ports de messages). Construit à partir des abstractions communes des modèles de composants de la littérature, ce modèle est conservé à l'exécution offrant une couche de réflexion à l'exécution sur lequel les outils de vérification et de simulations peuvent être facilement branchés. Les spécificités d'implémentation des composant sont effacées par le modèle de composant. Ainsi, les simulations et vérifications de conformité des assemblages peuvent être réalisées de façon uniforme. L'indépendance du modèle permet de mener ces tests sans conséquence aucune sur le logiciel en train de s'exécuter. Cette couche contribue à la sécurité du système, à la gestion de la variabilité et aux mécanismes adaptations en apportant des outils pour chacun de ces domaines. Des outils ont aussi été développés afin de garantir, en permanence, la cohérence entre les implémentations et leurs représentants dans le modèle.

Les **Wrappers** prennent en charge la publication des produits présents dans le système, sur des protocoles de niveaux applicatif comme DPWS ou UPnP. Ces publications automatiques ouvrent la solution à des protocoles applicatifs existants et futures, sans nécessiter une réingénierie complète du système. Souvent trop gourmands en ressources pour être directement embarqués dans les produits eux-mêmes, cette couche permet d'offrir gratuitement aux produits une présence sur ces réseaux applicatifs.

L'**Environnement d'exécution orienté services** complète la contribution en ap-

portant le support d'exécution du modèle de composants. Il donne vie aux capacités des différentes couches en supportant les adaptations et évolutions pendant l'exécution.

1.5 Adéquation de la contribution

Le contexte de l'aide à domicile de personnes âgées, la description du domaine de la domotique, et l'état de l'art, ont permis d'extraire une liste d'exigences identifiées comme essentielles pour qu'un système logiciel soit utilisable dans ce contexte. Le tableau 1.1 détaille les forces de chacune des approches considérées face à ces exigences. La dernière ligne du tableau présente la contribution de cette thèse, et montre ainsi ses forces comparées aux mêmes préoccupations.

		Interop.	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
Generic Approaches	OSGi [All11]		+	+	+		
	ESB [Cha04]	+	+		+		
	Darwin [GMK02]			+	+		+
	Koala [RvdLKM00]				+	+	+
	Fractal [BCL+06]			+			
	SCA [sca]		+		+		+
	FraSCAti [MRRS10]	+	+	+	+		+
	iPOJO [EHL07]		+	+	+		
Domain-Specific Approaches	uMiddle [NT07]						
	SOPRANO [WSO+10]			+	+		
	Gaïa [RHC+02]	+		+	+		
	Dia Suite [CBC10]	+	+			+	+
	Habitation [JRS+09]	+				+	
	WADL [CDT08]	+		+	+		
	PervML [MPC06]	+	+	+	+	+	
	AutoHome [BDLM11]		+	+	+		
	WComp [FHL+11]	+		+	+		
	Niagara [Tri08]	+	+				
	EnTiMid	+	+	+	+		

Table 1.1 – Adéquation de la contribution aux exigences

Dans la contribution de cette thèse, le problème de l'interopérabilité est la préoccupation principale de la couche d'interopérabilité des produits, assistée par la couche apportant le modèle de composant. L'ouverture est assurée par les *wrappers* au niveau des protocoles d'application, et par les *drivers* au niveau des constructeurs de produits. Les capacités d'adaptation pendant l'exécution et d'évolution sont rendues possibles par la couche de Model@Runtime et l'environnement d'exécution construit à partir d'une

architecture orientée services. Le contrôle distant est offert au travers des *wrappers* et du model@runtime. La gestion de la variabilité, quant à elle, est facilitée par la présence du modèle pendant l'exécution, mais les outils restent insuffisants pour résoudre complètement le problème.

1.6 Intégration à l'état de l'art

Au cours de l'étude de l'état de l'art et des approches, de bonnes propriétés ont été identifiées. La contribution de cette thèse est conservative par rapport à ces propriétés.

Le **modèle de réflexion indépendant** proposé dans le domaine de l'ingénierie des modèles, est rendu disponible par la couche *Model@Runtime*. Ce modèle abstrait de l'architecture pendant l'exécution est indépendant et synchronisé avec l'environnement d'exécution. Cette indépendance permet la création de raisonneurs capables de proposer des changements, et de vérifier la validité de ces derniers avant qu'ils soient réellement appliqués.
La **gestion externalisée des liaisons** entre les composants est imposée par la couche de Model@Runtime, et rendue possible par le modèle de composant, ainsi que par les *drivers* de la couche d'interopérabilité qui ne peuvent pas présumer d'une utilisation spécifique des composants. Cette explicitation des liaisons permet l'adaptation de l'architecture, ç-à-d la modification connexions, ou même les composants, pendant l'exécution. La nécessaire indépendance des composants pour permettre l'interopérabilité et les adaptations, renforce aussi l'intérêt d'extraire les dépendances.
Le **déploiement à chaud** fût conservé grâce au choix fait de construire les développements de l'environnement d'exécution sur une plateforme orientée services. Il était nécessaire de conserver cette propriété indispensable pour le support de l'adaptation, ou le faire évoluer, en déployant de nouveaux composants sans redémarrage.
L'isolation close entre les types de composants et les instances est aussi une mesure aidant à garantir l'interopérabilité, et la gestion indépendante des différents éléments composant l'application. En effet, les produits physiques ayant chacun des cycles de vie indépendants, il fallait que le système puisse les gérer de façon indépendante aussi.
L'ouverture, identifiée comme bonne propriété, le fût aussi comme une exigence incontournable pour ce type de systèmes. Ainsi, cette bonne propriété est aussi conservée.

1.7 Bénéfices immédiats

1.7.1 Simplification du développement de composants

Les outils développés en support de la contribution de cette thèse offrent une assistance au développement des composants. Le modèle de composant impose que les développeurs respectent certaines bonnes propriétés comme l'isolation close, ou l'externalisation des dépendances, ce qui rend la maintenance et les évolutions plus simples. L'extraction automatique du modèle des composants à partir du code, et les mécanismes

de synchronisation raccourcissent le temps entre l'expression des exigences et la fabrication de la solution, et préviennent de beaucoup d'erreurs.
En outre, la gestion du typage et les autorisations de connections entre composants étant traités au niveau du modèle, le durcissement(pour prévenir d'erreurs) ou l'alègement(pour permettre une connection particulière) des règles de vérifications se fait en ajoutant ou retirant des règles au niveau de l'éditeur, ou de la platefore d'exécution, sans necessiter de modification des implémentations.

1.7.2 Création d'applications par assemblage

Le modèle de composant et les outils de modélisation apportent un support à la création d'assemblages de composants, et donc, d'applications. Les librairies de composants créées par les développeurs peuvent être importées dans les éditeurs, afin que les composants soient intégrés et connectés. Les connections port-à-port permettent de connecter deux ports, quels que soient leurs noms ou utilités, si les outils de vérifications personnalisables valident ou non la pertinence de l'assemblage.
Par ailleurs, les développeurs de produits domotiques, familiers des composants électroniques, sont familiarisés avec le paradigme de composants de par sa proximité du modèle électronique. Cette rapide prise en main permet aux ingénieurs et techniciens de créer des applications personnalisées pour répondre strictement aux besoins de chaque personne.

1.7.3 Viabilité et précision

La possibilité de faire évoluer une configuration à chaud offrent au système la capacité de suivre les évolutions de la pathologie ou des technologies sans avoir à reconstruire le système complet. Ainsi, il est possible de créer un système logiciel finement adapté à des besoins à un instant, et de le faire évoluer par la suite avec un coût limité.

1.7.4 Intégration simplifiée d'objets et de services

Le modèle de composant proposé dans cette thèse permet la connexion de composants hétérogènes, non prévus pour fonctionner ensembles. L'hétérogénéité de ces composants peut être due à leurs constructeurs, aux protocoles qu'ils utilisent, aux média de communication, mais aussi due à l'objet qu'ils représentent. Plusieurs services rendus à travers Internet ont été incorporés, enveloppés dans des composants, et permettent par exemple, à une application d'accéder à un agenda en ligne ou un service de météo. Une gestion horaire de l'allumage d'un éclairage peut par exemple être défini en connectant le composant gérant la lumière à celui d'un Google Agenda. Dans ce cas, un rendez-vous dans l'agenda symboliserait la période d'allumage de l'éclairage.

1.8 Limitations identifiées

1.8.1 Description structurelle

Le modèle de composant proposé dans la contribution de cette thèse facilite la description structurelle de l'architecture d'un système, alors que les gens sont plus enclin à décrire le comportement qu'ils attendent d'un système logiciel. En plus du modèle de composant (et donc de la description structurelle de la solution), un outil permettant de décrire le comportement attendu de l'assemblage devrait être proposé. Dans ces conditions, un utilisateur final pourrait être capable de modifier le comportement du système sans avoir à s'occuper de la structuration des composants internes.
Le principal challenge réside dans le fait que le comportement global du système peut être décrit en plusieurs morceaux. En effet, les gens seront capables de décrire ce que le système doit faire si une alarme se déclenche, si la porte s'ouvre ou s'il fait froid, mais n'auront pas une vision globale du comportement du système, et des conséquences de chacun de ces petits comportements. Les différents comportements peuvent par ailleurs interagir les uns avec les autres et amener le système dans des états incohérent.
Enfin des utilisateurs non experts du domaine, et les utilisateurs finaux ne le sont pas a priori, n'expriment que le comportement nominal attendu. A partir de cette description, des outils doivent analyser les points probables de défaillances ou d'erreurs, et tenter d'y parer.

1.8.2 Paramètres de ports

Les mises en œuvre de différents modèles de composant classiques ont été exclus des solutions envisageables pour cette thèse, parce que la spécification de leurs ports par une interface logicielle était trop stricte pour le domaine considéré. Trop stricte, parce que les méthodes et les paramètres de ces méthodes, s'ils ne sont pas parfaitement alignés, empêchent toute connexion sans l'intervention de connecteurs *ad-Hoc*. Le modèle de composant proposé dans cette thèse n'a pas complètement résolu ce problème. Il a été remonté au niveau du modèle, le rendant plus simple à gérer.
Si le problème d'alignement des paramètres n'a pas été vraiment traité dans cette thèse, il a cependant été identifié et des solutions apparaissent dans la littérature scientifique par l'utilisation de connecteurs [MB05], pouvant avoir des comportements plus complexes qu'un simple appel de méthode ou envoie de message. Des mécanismes de *mapping*, de synthèse de connecteurs ou de re-nommage, par exemple, peuvent être imaginés pour résoudre l'alignement [CBJ10].

1.8.3 Des vérifications basiques

L'expérimentation n'a pas nécessité la mise en œuvre de vérifications complexes des modèles. Seulement quelques vérifications structurelles sur le modèle ont été implémentées pour rechercher des cycles de dépendances bloquantes pour la gestion du cycle de vie par exemple. Beaucoup des points de vérifications n'ont pas été complétés, parce qu'ils sont dépendants de règles métiers. A chacune des étapes de vérification, les outils

s'intéressent à différents aspects de la validation de l'application, et n'ont donc pas les mêmes besoins d'information pour prendre leur décision. Comme il est probable que le modèle ne contienne actuellement pas toutes les informations nécessaires pour chacune des étapes de vérification, l'approche vise à permettre l'extensibilité de ce modèle par ajout de méta-données nécessaires à la vérification de propriétés.

1.8.4 Gestion de la variabilité

Le problème de la variabilité n'a pas été complètement adressé, parce qu'un petit ensemble de composants était suffisant pour tester les fonctionnalités dans le cas général. L'expérimentation, elle aussi, a été réalisée à partir d'un ensemble de composant qui a pu être géré. Dans la perspective d'un déploiement réel, les variations des configurations vont imposer la création d'outils d'aide à la gestion de ces variations. Des réflexion utilisant le développement à base d'aspects, les lignes de produits logiciels et l'ingénierie des modèles pour traiter la question de la variabilité ont déjà été menés dans le cadre du projet DiVA par exemple [MFB+08].

1.8.5 Absence de test sur des plate-formes embarquées

Dans le contexte de l'aide à domicile de personnes âgées, le choix a été fait de conduire les expérimentations sur un ordinateur tout-en-un, équipé d'un écran tactile. Ce PC était doté de capacités de calcul et de mémoire supérieurs à la plupart des plate-formes embarquées. Dans un déploiement en environnement réel, il se peut que l'écran tactile ne soit ni nécessaire ni souhaitable pour remplir des fonctions d'automatisation, et qu'une plateforme d'exécution embarquée suffise. Des études en cours tendent à montrer la validité de la solution sur des plateformes ARM munies d'un Linux embarqué et d'une machine virtuelle embarqué comme *JamVm* ou *Oracle Embedded*.

1.9 Contribution au réseau d'excellence Européen S-Cube

1.9.1 Le réseau d'excellence S-Cube

Cette thèse a été menée dans le cadre du Réseau d'Excellence Européen S-Cube. S-Cube [1] est un réseau d'excellence Européen en Logiciels, Services et Systèmes. Ce réseau d'excellence a pour ambition de coordonner la recherche européenne sur les services logiciels. En connectant la recherche à l'industrie, et en unifiant des recherches multi-disciplinaires, S-Cube cherche à développer des méthodes d'ingénierie des services, agiles et globales, et spécifier les principes et techniques d'adaptation des services.
Ce réseau d'excellence a été financé par le programme de recherche Européen (FP7) 'Coordination', sous le thème des

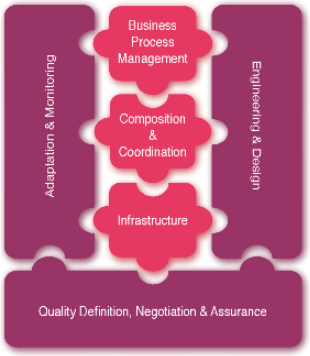


FIGURE 1.1 – S-Cube Research Framework

1. http ://www.s-cube-network.eu/

Technologies de la Communication et de l'Information (ICT). En plus d'offrir un fort support pour les collaboration et des opportunités de mobilité entre les instituts de recherches Européens, S-Cube a financé plusieurs thèses de doctorat pour les différentes couches de la vue globale présentée en figure 1.1

1.9.2 Contribution

La contribution de cette thèse s'inscrit dans les travaux du groupe de travail 1.2 : *Adaptation and Monitoring Principles, Techniques and Methodologies for Service-based Systems* de l'activité de recherche collaborative(JRA) 1 : *Engineering and Adaptation Methodologies for Service-based Systems*

L'objectif général du JRA1 est de "concevoir un ensemble de principes intégrés, de techniques et de méthodologies pour la conception, l'adaptation et la surveillance d'applications basées sur les services, tout en garantissant la qualité du système de bout en bout, et sa conformité au contrat de service", d'après la description du travail d'S-Cube [2].
La contribution de cette thèse comporte un modèle de composant qui : implique de nouvelles méthodes et techniques d'ingénierie, permet l'adaptation d'applications basées sur des services, et offre des moyens de réaliser des vérifications de la conception au déploiement pour assurer la qualité de service.

Plus précisément, la contribution de cette thèse s'inscrit dans le groupe de travaille JRA-1.2, qui cherche à définir de nouveaux principes et techniques pour l'adaptation et la surveillance d'applications orientées services, globalement sur l'ensemble des couches. Si EnTiMid ne s'intéresse pas à priori aux problématiques de surveillance, il permet de résoudre les questions d'adaptation.

Du point de vue du projet S-Cube, EnTiMid peut aussi être considéré comme une brique logicielle permettant l'adaptation des couches d'infrastructure ou de composition et coordination de services, couches présentés sur la figure 1.1.

2. DoW Amendment 4, December 6th, 2010

Part I

Context, Requirements and State-of-the-Art Review

Every person I work with knows something better than me.
My job is to listen long enough to find it and use it.
Jack Nichols

The European population is getting older due to a conjunction of two factors. First, the decrease of births reduced the part of youth in the population. The second factor is the soon arrival of post-war "baby-boom" people to the age of retirement. Both of these factors imply a radical change in the age pyramid, and in the socio-economical environment of European countries. A consequence of this ageing of the population is an emergence of needs and requirements to face this global evolution.

Over the past few years, home automation technologies have been tending to democratize. More and more technical solutions are proposed to automate shutters, garage doors or lightning in houses. These facilities improved the quality of life of the European population. Now they sound like an interesting tool that could help and offer support to elderly people in their home.

As an introduction, chapter 2 presents the Ambient Assisted Living and Home Automation domains, in order to extract some general requirements and outline the contribution of this thesis.
After this introduction, a state-of-the-art review in AAL projects, Home Automation, and software engineering approaches is carried out in chapter 3. Chapter 4 ends this first part with a summary of the state of the art, and announces the contribution of this thesis.

Chapter 2

Introduction

Home Automation and the Ambient Assisted Living(AAL) domains have been of major influence on this work. Home Automation technologies offered a plethora of technical solutions with various constraints, while AAL brought substantial real life material in terms of requirements, needs, or use cases.
This introduction chapter presents these domains in sections 2.1 and 2.2. This presentation enables section 2.3 to list some general requirements identified in these domains. Section 2.4 defines the scope of this work before section 2.5 outlines the contribution of this thesis.

2.1 Ambient Assisted Living

2.1.1 The origins

According to Eurostat [1], the median age of European Union (27 countries) population has been growing regularly. From a median age of 37.7 years in 1999, it increased to 40.6 years in 2009 as shown on figure 2.1. It is a fact; the European population is getting older each year. This ageing of the population is the result of the combination of several factors, among which are the ageing of baby-boomers, and the decrease of birth rates.

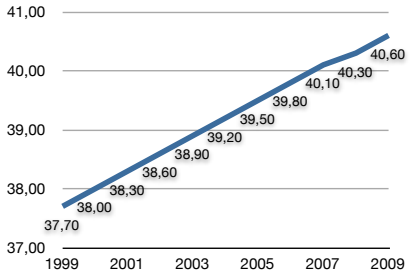


Figure 2.1: Median Age of EU Population - Source Eurostat

The **"Baby Boom"**
During the Second World War the birth rate stagnated, resulting in a similar number of births from 1939 to 1945. This stagnation is visible in figure 2.2, at the level of people aged between 64 and 70. The "Baby Boom" describes the rapid and strong increase

1. http://epp.eurostat.ec.europa.eu/portal/page/portal/eurostat/home

in the number of births that occurred after the Second World War, thus between 1945 and 1968. Actually, 4.9 million people were born in 1944 in the EU, 7.6 million were born in 1968 (+35.8%). People born during the "Baby Boom" are now (in 2011) 43 to 68 years old, and will soon retire.

Decrease in birth rates
As can be noticed in figure 2.2, a decrease in birth rates began at the end of the sixties. From 7.664 million persons born in 1968, the number of births fell to 5.061 million in 2002 (-33.4%). In [Bos98], Xavier Bosch explains that this phenomenon is due to a multitude of factors such as an increase in the use of contraception, the raise of the number of single people, or the increase in the percentage of women in the workforce.

Europe will soon have to face the increase in the retired portion of the population, and a simultaneous decrease in young people entering working life. By 2050, the number of people over 65 in the EU will have increased by 70%, and the number of people over 80 will have grown by 170 %.

In order to be ready on time, governments must address the economic and social implications of an ageing population. They must prepare for increasing demands on healthcare, as a rapidly ageing society heralds growing populations with chronic diseases, disabilities, and increasing health needs.



Figure 2.2: Age Pyramid EU (27) in 2009. - Blue: M, Green: F - Source Eurostat

2.1.2 The concept

The Ambient Assisted Living Joint Programme [joi] defines the concept of Ambient Assisted Living (AAL) through 6 dimensions.
Autonomy By increasing the autonomy, the self-confidence and the mobility of elderly people, AAL tends to extend the length of time people can live in their preferred environment.
Activities Maintaining physical or intellectual exercise helps elderly people to remain in good health, and prevents a decrease in capacities.
Assisting individuals at risk, by promoting a better and healthier lifestyle.
Securing support and maintaining the network around the individual, including family, friends and social activities, to enhance security and prevent social isolation.
Supporting carers, families and care organizations in their everyday activities.
Streamlining the use of resources dedicated to elderly people, by increasing their effi-

ciency and productivity.

There are many solutions to address these dimensions. Automation of some tasks can enforce autonomy, and the use of mechanical aids can improve mobility. Social workers and health professionals can propose activities, support and assistance. Unfortunately, healthcare associations or companies have difficulty in hiring people for these jobs. Indeed, qualified people are not numerous enough, and financial constraints are strong in this domain.

A solution, in assisting both helped people and helpers, could be to use Home Automation technologies in conjunction with ICTs and human interventions. Section 2.2 presents this domain.

2.2 Home Automation

In 1962, William Hanna and Joseph Barbera created the Jetsons cartoon family. In this carton, George and Jane were living in the Skypad Apartments in Orbit City with their children Judy and Elroy. Their housekeeping robot, Rose, handled all chores not done by the numerous automated appliances triggered with some push-buttons. Apart from the fact that it is just a fiction cartoon, it describes well the idea of smart homes or home automation. Let us get into a bit more detail about home automation, and see why it is still not applied in everyone's home.

2.2.1 Application domains

Home automation has for too long been perceived as it was presented in the Jetsons: a set of costly, useless, funny pieces of technology.
Personal home theaters, multi-room media systems, smart colored lighting controllers will always exist and make a good showcase of what is possible. Besides, the use of more utility-oriented home automation is soaring. Home or building automation technologies, among others, ease the management of lighting control, shutter control, heating, ventilation, air conditioning, energy management, metering, monitoring, alarm/intrusion systems, household appliances, audio/video and lots more. However, people are not ready to pay for the automation of tasks they can execute by themselves.
Widely used in industry buildings and plants, the automation of the lighting or heating systems has brought substantial savings of energy and money. This industrial experience makes it possible to imagine the benefits of installing such systems in homes. The minimization of power consumption with a maximization of welfare is acceptable, even dreamed of by everybody. By extension, alarm systems, automatic garage doors and shutters can also be managed this way, rendering a global and coherent service to inhabitants.

2.2.2 Technologies

The devices encountered in home automation originate from different business domains. M. Nagy says in [MAO+09] that "A major problem is inherent heterogeneity [...] with respect to nature of components, standards, data formats, protocols, [...]". Indeed, it can be a problem from many aspects, but it is also a set of tools waiting to be connected.

Communication Media

Four main communication media can be differentiated in home automation technologies: the *Bus*, the *Radio*, the *Power Line Communication (PLC)* and *Infrared*. Devices use these communication media to communicate with each other, and offer a technical functionality. The choice of a medium is linked to the constraints to be addressed. For instance, a Bus link is highly reliable, but requires the wiring of the entire house. On the other hand, Radio communications do not need any additional wiring, but are less reliable and batteries have to be changed regularly.

Plethora of protocols

Historically, there is no specific home automation brand or manufacturer. It all comes from different trades such as electricity or Heating Ventilation Air-Conditioning (HVAC) control. According to the specific needs of their domains, each home automation manufacturer has developed a specific communication protocol for its devices to operate with each other. They also use several communication media to carry the communications. A consequence of that is the huge number of devices, protocols and media available on the home automation market, illustrated in section 3.1.4. Technical solutions proposed by the Home Automation domain are so numerous, that there may be a solution for each need, in almost each domain. Nevertheless, due to this great diversification, no common management tool exists so far, resulting in expensive mono-manufacturer closed solutions, or no solution at all.

Even if home automation technologies have existed for many years, no real standard has been released. Imposing a global standard, as was the case for the IP protocol for example, seems to be the best way to address such an issue. Each domain has created its communication protocol adapted to each business concern. Thus, finding a common protocol, used and understood by all devices from all domains, appears to be a very difficult task. And even if one finally emerges, developers will still have to deal with legacy devices using proprietary protocols.

Moreover, the old way manufacturers think adds to the complexity of the problem. They still think that a closed world (I mean non-public protocol specifications) is a world that can be controlled. Indeed, it is true, but it is also a world that fewer and fewer people want to enter, because they are afraid of the captivity imposed by such solutions. Captive systems are often well tested, because of the restricted number of available devices, but there is also a risk for these components of being removed from the market one day. In these conditions, people could not replace these components with compatible others, and are thus really concerned with the sustainability of the adopted solution.

2.3 Identification of requirements

The domain of Home Automation requires a new breed of technology to easily manage installations and answer specific needs. The lack of such a tool, able to operate any home automation technology and create solutions for specific needs of elderly people, makes the adoption and use of these technologies marginal. This absence of a tool may be due to the complexity and number of requirements inherent to home automation systems. This section aims to identify a set of required properties for a software tool to be adopted by manufacturers, installers and users.

The requirements presented in this section have been identified in [NBFJ09]. This section completes the list and elaborates on them.

Interoperability

Interoperability is described as the ability for systems to operate with each other. Even if 'system' is quite a generic word and encompasses lots of things, the idea behind interoperability is simple. Given two or more systems, how can one ensure that each of them is operable with another one? For instance, if I have a light management system and another one for shutters, how can I guarantee that systems will be able to communicate in order to be of use for the user?
A closed environment, in which all devices come from the same manufacturer, avoids the problem. A communication interface, common to all components, also deals with the issue. Interoperability becomes a complex problem when the environment is open and when several technical concerns have to be handled in a single place (lighting and heating management for instance).
In the context of AAL, each solution will be uniquely deployed, because each patient, each helped person has specific needs due to his environment or illness. Indeed, specialists in this domain will select different technical artifacts or services, according to each person's needs. Those solutions will have to cooperate with each other in a single system, to perfectly meet the user's needs, whoever their manufacturer or whatever their communication protocol may be.

Openness

Openness means making all offered functionalities available for third party applications, or different uses other than the one initially imagined. In closed systems, customers are limited to the functionalities and evolutions proposed by the solution's manufacturer. If the system deployed was built specifically for a customer, any evolution is very costly. Keeping the system open to external contributions may reduce the number of demands for costly system modifications or add-ons. In any case, it is also an open door for new unforeseen appliances adding smart behavior to a reliable set of functionalities.
This is a clear challenge in computer sciences, and in general when considering com-

munications and interactions between objects or systems. For instance, Melison et al. propose in[MRRS11] to extend the Service Component Architecture (SCA) specification to introduce new bindings to handle new communication for pervasive or social use. This ability of extension opens the solution to future evolutions. The definition of a home-made interface fulfilling the requirements is easier than fitting into a good practice or a standard that does "almost what we want but not completely".
Openness is a strong requirement in home automation systems, as it has been in computers. Today, a computer manufacturer would not create a specific connection port, compatible with no other computer, unless he wants to create a new captive market.

Adaptation
In a perfect world, devices never fail, services are always available and the Internet is always available. As demonstrated every day, the world we are living in is not perfect. Software systems, or systems dealing with objects or services linked to everyday life actions, have to take their execution environments into consideration. They should be able to dynamically adapt to changes around them while they are running, in order to maintain a given level of services or functionality as long as possible. These adaptations should not require any reboot of the system, because a reboot could make all in-house functionalities unavailable. Lastly, an adaptation is not intended to add or remove any system functionality permanently. Otherwise, it is an evolution.
Execution policies or system reactions must be easily modifiable, in order to take into account any change in the user's requirements. For instance, depending on the level of dependency of an elderly person, the system may completely, partially or not at all automate the management of certain household functions such as heating or shutter management. This type of change in system behavior has to be made simple and operable by any authorized person, even with no specific skill in this domain.
Most of all, these changes during runtime must not affect the execution of basic security functions (such as emergency call handling), neither in their behavior, nor in their execution during adaptation.
Adaptations do not only concern technical elements, but also the user himself. All users are different. While some may be interested in new technologies, others are completely agnostic. When some are able to remember and learn how the system behaves, others may experience some memory losses. Where some have vision disorders, others experience hearing limitations. Software systems should be able to take into account these disabilities and adapt to their evolutions and to users' requirements.

Evolution
Evolution in the context of home automation and/or assisted living is a key requirement. Needs or uses are changing, protocols and technologies also. Some functionalities may finally be required, whereas others can become useless and need to be uninstalled. Security or communication protocols can be improved and deployed in new versions that have to be taken into account without needing to re-implement the entire system. Moreover, systems deployed in a house or a building, in charge of its management and the comfort of its inhabitants, have to be designed to serve during the entire lifetime

of the building (even if hardware changes may be required). Therefore, they must be ready to accommodate future and unforeseen evolutions such as the installation of new services/functionalities for example.

Variability Management

No house or building is like any other. This is because of structural or ground specificities, because of a particular user's need or just because no one would like to have exactly the same house as his neighbor. Thus, each installation will have specific devices deployed, using specific versions of protocols, and providing several functionalities. The development, by hand, of a specific control system for each installation is completely out of the question. It would be too costly and error prone. A global management tool should exist to deal with the inherent variability of such systems.
People responsible for designing a solution for an installation query should have some tools at their disposal to help them in choosing devices to install, or to assist them in selecting devices from all the catalogs of all manufacturers.

Remote Control

More and more, people want to remotely access their belongings. This is also true for their homes. They would like to be able to remotely check the lights status, run a bath or switch on the alarm system that they forgot.
In the context of building management facilities, specialized companies have different solutions to check the state of a building system. They either have remote access to all systems, or they check locally by hand. The local solution is no longer viable in the context of our society. Systems deployed to control buildings or homes should have remote access possibilities (under access control and agreement) to allow for remote diagnosis, corrections, or evolution actions from a control center.

Distribution

Today's systems work more and more with or on different execution platforms, and communicate more with each other. This is particularly true for home automation. For redundancy reasons and service level insurance a building and even a house can be equipped with multiple, independent, but connected controllers. These controllers can share the different functions to balance the loads on platforms, or offer a connection to a specific, locally-connected device. When a running platform fails, other platforms aware of its job can stand in temporarily, until the original controller is back in operation. Also, devices are becoming smarter and may make decisions by themselves. As a consequence, decisions and control could be distributed to several smart pieces of equipment.

Safety & Security

As presented in the context of this work, safety and security are very important requirements for home automation systems. Indeed, they themselves have to be safe and secured to be able to play their role in difficult situations and improve the security and safety of people and goods. A minimum service level has to be guaranteed, for

inhabitants not to remain stuck in the house in the case of an emergency. Moreover, the access policies of the system have to be sufficient to avoid unauthorized access, and simple enough not to become a constraint for authorized carers.
Several tools such as simulators, tests, or model checking, can be used to check and improve the reliability and the safety of such systems.

Acceptability & Accessibility
In the domain of home automation, deployed solutions, whatever their nature, have to take into consideration interactions with all inhabitants. Whether they are children, teenagers, young workers, parents, retired or old people, all of them must be able to interact with the system. Checks have to be performed on all solutions proposed, to ensure all users can control, or obtain information from the system and keep control.

New requirements for software are emerging from the accessibility of home automation technologies. The evolution of the software during the building's lifetime, its adaptation to cope with changes in its execution environment and the huge variability of technologies and protocols to be guaranteed to inter-operate, are triggering new challenges for software engineering.

2.4 Scope of this work

The list of requirements presented in the previous section could be completed. A particular situation can bring many other issues for software engineering.
This thesis focuses only on a subpart of these requirements and aims to propose a solution to conjunctively address interoperability, adaptation, openness, evolution, variability management and, partially, safety and security issues. The first goal of this thesis was to end up with a working solution able to cope with this sub-list of requirements. Other ones will be addressed in future works. As for safety and security issues, the work realized in this thesis partially addresses the problem, since questions about privacy and information security concerns have been decided as out of scope. The focus was more about ensuring consistency of assemblies and a fail-safe system.

2.5 Contribution of this thesis

Software can no longer be built on a once-off basis. Customizable, reliable and personalized solutions have to be deployed in the short terms to meet changes in users' needs or in the environment at any time. Moreover, the specificities of each installation make it very hard to create a unique piece of software able to cope with all technologies, concerns (such as energy consumption) or unpredictable evolutions.

In the electronics domain, the number of components and their always-possible connectivity have offered technicians and engineers the means to create various solutions. Even many years after their assembly, electronic devices can still be repaired or completed with new features. The proposal made in this thesis is to take advantage of the electronic way of doing to improve the flexibility of software systems while keeping a high level of safety and security.
To this end, the contribution of this thesis can be described from three aspects:

- A constraint-relaxed component model that leverages the software flexibility, by offering ways to connect any component to any other. This aspect addresses interoperability issues and evolution requirements

- Modeling tools to create, modify and simulate component assemblies, check their consistency and validity before their (re-)deployment at runtime. Safety and security, as much as variability management are requirements covered by this aspect of the contribution.

- An execution environment built over a service-oriented runtime, to support the proposed component model and cope with adaptation and evolution requirements at runtime.

Chapter 3

State-of-the-Art Review

The review proposed in this chapter is made if three sections.
Section 3.1 starts with a description of the Ambient Assisted Living domain from which social requirements have emerged, and it presents some projects on this topic. It then introduces some home automation technologies, their use and goals in several projects. The overview of these two domains makes it possible to sense more precisely the needs for a new breed of software for these domains.
Section 3.2 then presents several general purpose software approaches and tools that have been considered for the resolution of the problem targeted. Each of them is evaluated against the requirements identified in the introduction. However, general purpose approaches seem not to be specialized enough to meet the requirements of the AAL and home automation domains.
For this reason, section 3.3 evaluates some selected domain-specific approaches with respect to the same requirements as for general purpose ones.
Finally, the review made in this chapter is summarized in chapter 4 which uses this basis to specify the contribution of this thesis.

3.1 Background on AAL and Home Automation

Ambient Assisted Living is a hot topic in Europe. Several projects have been led to address different aspects and try to cover the needs inherent to a home-keeping situation. This section introduces some of them.

3.1.1 Projects in AAL

The Ambient Assisted Living (AAL) Joint Programme [1] is a collaborative association of twenty European Union Member States, plus three Associated States. They group together the AAL Association, whose main objective is to enhance the quality of life of elderly people, through the use of Information and Communication Technology (ICT). Their main activity is to found R&D projects in the AAL domain and to publish

1. http://www.aal-europe.eu

annual calls for project proposals.
The total budget planned for the AAL Joint Programme is €700 million, half of which is to be funded by public resources (AAL Partners plus the European Commission) and the other half by private participating organizations. According to the 13th article of the EC decisions 743/2008/EC [otEU], funding from the European Commission is taken from the budget allocated to the ICT theme of 'Cooperation' under the FP7 Specific Programme of the same name. Thus, the AAL-JP is set to last from 2008 to 2013, to correspond to the FP7 dates.

The **ASK-IT** project [2] ran from 2004 to 2008, and aimed at providing *Ambient Intelligence* to support and promote the mobility of impaired people [WSV+07]. It laid to the development of a software framework, and a set of tools for mobility. Among other things, a *Domestic Module* has been created to support the provision of seamless home environment management, to the mobility-impaired traveler on the move.

In a slightly similar approach, the **SOPRANO** project [3] intends to design an ICT system to foster the comfort and safety of elderly people in their everyday life. In this project, a strong effort is made to measure the acceptability of the solutions. SOPRANO [SML+10] aims to maximize the acceptability of the services, especially in populations vulnerable to loss of independent living. This is achieved by ensuring control of the environment by users, where they so wish and by using extended user-centered design techniques.

INHOME IST Project [4] is also an interesting project in this context. Completed in December 2008, its goal was to enhance the autonomy and the safety of elderly people at home [VAMC08]. It led to the development of a set of services such as activity monitoring, home environment management, task scheduling, flexible AV stream handling, and hence, flexible access to household appliances.

As for, the **GUIDE** project [5], it endeavors to provide a framework [CCQS05], a user model toolbox and a handbook, along with graphical user interface components, in order to ease the creation of graphical user interfaces answering the special needs of elderly people.

All these projects try to improve the comfort, safety and security of elderly people by means of technical aids. Home Automation is a vast field that offers many solutions to ease or help in the realization of everyday tasks that can become difficult with ageing. Basic elements of this domain have been introduced in section 2.2.

2. http://www.ask-it.org/ (march 2011)
3. http://www.soprano-ip.org/ (march 2011)
4. http://cordis.europa.eu/fetch?ACTION=D&CALLER=PROJ_IST&RCN=80489 (march 2011)
5. http://www.guide-project.eu/ (march 2011)

3.1.2 European research

The European 'Cooperation' Programme of the FP7 is quite a good representation of European countries' current concerns and research themes. According to the FP7 factsheet[Com], €32.365 million are allocated to different themes. The major theme, which is allocated 28.72% of the FP7 'Cooperation' funds, is ICT. In second position with 19.07% comes the health theme. Third and fourth positions are Transport and Nano-Productions respectively, with 13.18% and 11.03%, followed by the Energy and Environment themes consuming 7.25% and 5.67% of the overall budget.
Thus, three of the top five concerns of Europe are ICT, Health, and Energy and the Environment (assuming Energy and the Environment can be grouped as a unique theme).

Using ICT to improve the quality of life of people is an idea that can be identified in the background of several projects throughout the world. Section 3.1.3 presents some of them with a focus on those using home automation technologies.

3.1.3 Home automation in projects

Information reported in this section has been partly collected from a talk by Luc Balanger, director of the Communication Networks department at FFIE (French professional association of electrical engineering companies).

Asian countries have developed strong market sensitivity to video games. Some TV channels are even specialized in the live transmission of gaming parties, involving professional and sponsored gamers. Over the past few years, several studies and news programs have reported a kind of addiction of a portion of the population to video games. In particular, A. Faiola in [Fai06] states that about 2.4% of 9 to 39 year-old South Koreans are believed to be suffering from game addiction, according to a government-funded survey. Another 10,2% of them were found to be obsessed with playing electronic games.
Some home automation manufacturers are working on products to make the game more real. The goal is to give gamers the sensation of being in the game, and playing as a first person, using for instance 7.1 speaker systems or 3D visualization devices. This aspect of home automation is clearly entertainment oriented.

The United States is also concerned by video game addiction of youth. However, home automation does not target this domain, but a much more prominent one in the US. The safety and security of people and goods has been a huge market in the United States since the 9-11 events, as explained by Terrell E. Arnold, a retired Senior Foreign Service Officer of the United States Department of State in [Arn]. Also called the "fear" market, American people are investing a lot of money to feel safe. In this field, video security cameras can be deployed in homes, for inhabitants to be able to remotely see what is happening. Presence simulation is a must in this kind of systems, feigning somebody's presence in the home by automatically switching on and off lights when

inhabitants are away.

In **Japan**, safety and security are also two important requirements and necessities, but not for the same reasons. As recently demonstrated, nature is not kind in Japan. Volcano, earthquakes and tsunamis are real dangers. The JEITA House Project [6] made it possible to automatically secure the house when an earthquake is detected. For instance, the solenoid valve controlling the arrival of gas in the house is switched off when an alert is received. This security has been enabled by the use of home devices able to communicate with each other, or with the centralized house manager. The JETI project also had an orientation toward improving the comfort of people. Asian people are fond of multifunction toilets or showers, and more generally, pay a strong attention to their well-being and health. A centralized house manager can improve the well-being of inhabitants by making everyday devices smarter. Manufacturers in this domain design their products to be more and more connected, and full of high-technology features. Moreover, each device is given a specific address making it possible to remotely control almost everything in the house. For example, it is possible to remotely run a bath using a smart phone for it to be ready when the resident is back home.

Home automation has a bad reputation, due to several advertisements that promoted useless functions. It thus has been considered as a costly, useless technology, for people fond of high technologies. Nevertheless, home automation technologies have quietly grown, and are today very rich in terms of communications, functionalities and uses.

3.1.4 Home Automation details

The evolution of concerns and needs in the home automation domain has led manufacturers to adapt their products over the years. As a consequence, lots of products communicating through many different media are available today. According to their manufacturers, and to their domains of use, devices also come with their own transportation protocols making their diversity even greater. This section briefly presents a non-exhaustive list of protocols and media used by home automation products to highlight the complexity of this domain with relation to the huge variability of solutions.

3.1.4.1 Communication Media

Communication Bus The bus is physically a wire, a link between devices responsible for transporting data packets from a source to a destination. When a data packet is sent, every device connected to the bus receives the packet. Most of the time, devices ignore the packets, unless the destination address of the packet is the device's address. Ethernet is probably the most famous ambassador of this communication medium.

6. http://www.eclipse-jp.com/jeita

PLC is a special kind of bus. The main idea of this communication medium is to reuse existing cable infrastructure to avoid adding a new specific communication cable. As the most common cable present in all houses is the power line, this technology injects data into the power line by modulating in a carrier wave. All transceivers plugged into the line can then decode the data from the carrier wave modulations. This communication medium tends to be more and more used, as it offers great data transfer rates and does not impose any new wiring.

Radio The radio medium is widely used, thanks to its wireless property. Radio devices are quite complicated to design, because of the trade off between energy consumption and protocol reliability. However their deployment is easy, and does not require a major work. Many manufacturers use the free ISM radio band. As a consequence, the ISM frequencies are noisy, and protocols have to secure their communications.

Infrared The infrared communication medium is employed for specific appliances. Indeed, there can be a lot of noise coming from the natural light disturbing the receivers. Since this medium is based on an optical link (using infra-red), both transmitter and receiver have to face each other. This is a strong limitation that makes the use of this medium difficult in home automation. In fact, devices are rarely facing each other. Nevertheless, it still is a good medium for a human-computer interface such as a remote control. Users can give orders to the system using a remote control, thanks to receivers disposed in many places around the house.

Links between media It is sometimes necessary to use a combination of different media to cover requirements from both the users and the infrastructure of the solution. As a consequence, manufacturers have developed families of devices, which use several communication media. They also created products to transfer orders/communication frames from one media to another, so that all their products can obtain the information.

3.1.4.2 Transport Protocols

io-homecontrol® is a two-way radio technology and a proprietary protocol. Organized in an association, a dozen of industrial manufacturers provide products compatible with io-homecontrol. Partners of this association are Honeywell, Niko, Somfy, Velux, Groupe Atlantic, Assa Abloy, Ciat, Renson, WindowMaster, SecuYou and Overkiz. This technology is embedded in house equipment such as roof windows, vertical windows, roller shutters, gates, garage doors, front doors, alarm systems, lighting, ventilation, heating systems, etc. A single control can monitor and pilot all the io-homecontrol compatible equipment in the house.

IOBL In One By Legrand is a proprietary protocol used by Legrand (a French electrical component manufacturer). Legrand offers many devices able to interact with each other. These devices control lights, shutters, or heating systems. Being proprietary, no

other device manufacturer offers compatible products. IOBL products are able to send and receive orders from infrared medium or PLC.

X2D Delta Dore is historically a French heating system manufacturer. This company has extended its activities to home automation and alarm systems, for home and building control. For their products to communicate, they have developed a communication protocol called X2D, using radio, bus and PLC media. Just as IOBL, the protocol is private, and no other manufacturer offer compatible products.

X10 is a communication protocol for low cost home automation products. Sensors and actuators send data frames over PLC or radio to a maximum of 256 addresses, with no acknowledgement. It is thus impossible to know if an order has reached its destination, and there can be only one order at a time on the network.
This protocol is quite limiting in term of number of devices, and guarantee of service, but it works and is pretty cheap compared to other product families. Moreover, technical specifications are available.

Z-Wave is a wireless technology to remotely control home appliances, entertainment products, and access systems, running in the 868MHz ISM band. Grouped into the Z-Wave Alliance [7], 160 manufacturers offer interoperable products in these domains, using this closed protocol. Based on a mesh topology, Z-Wave data frames can transit through several devices around the house to reach their destination. This is an interesting faculty to overpass radio obstacles and ensure delivery.

ZigBee is an open standard addressing low-cost, low-power M2M wireless networks. It was released by the IEEE in 2003 and over 300 manufacturers have since then joined the ZigBee Alliance. Working on 2.4GHz, 900MHz and 868MHz, ZigBee has been designed to be able to work in hostile radio environments. Network topology can be point-to-point, infrastructure or mesh, and accepts up to 65,000 nodes. Wireless products compatible with the ZibBee specifications target remote control, home automation or sensing domains.

6LoWPAN is the name of a working group at the Internet Engineering Task Force (IETF). This group aim to reduce the memory footprint of IP frames (and principally IPV6 frames) for the protocol to be used in devices and networks with low power availability. This protocol would make it possible to use the powerful IP routing mechanisms, in sensor networks or distributed embedded applications, making them operable from any classical IP network. Specifications of 6LoWPAN can be found as RFC 4944 [8] and RFC 4919 [9].

7. http://www.z-wavealliance.org/
8. http://tools.ietf.org/html/rfc4944
9. http://tools.ietf.org/html/rfc4919

HomePlug is an alliance of industrial companies. The goal of this alliance is to enable and promote the use of PLC networks and products. Industrials involved in this group take places at each level of the value chain, from service and content providers to software/hardware design. This alliance is and has been involved in several IEEE Standard definition processes. HomePlug took part in the definition of the HomePlug AV [All05] (IEEE 1901.2010), tend to launch a new certification program for the Low-Frequency Narrow Band Powerline Communications standard [10] (IEEE P1901.2), and support the elaboration of the IEEE P1905 standard which aims at providing a common abstraction layer to several existing home network technologies.

Dash7 Alliance aims to promote the adoption and use of the ISO 18000-7 [11] standard. This group of 20 members (as of Nov. 2011), encourages the development of compatible products, educates the market to this technology and certifies products capabilities. Dash7 is a wireless long range and low power technology. It supports IPv6, P2P messaging and encryption using 128-bit AES or public key. Dash7 operates in the 433 MHz ISM band with a dynamically adaptable data rate between 28 Kbps and 200 Kbps.

European Eib/KNX consortium The KNX Association [12] was founded in May 1999, by the members of the EIBA (European Installation Bus Association), EHSA (European Home Systems Association) and BCI (BatiBUS Club International) associations. Its mission is to develop and promote the KNX standard so that it is recognized as "The worldwide standard for home and building control". The KNX standard has been designed to enable the control of applications in industrial, commercial and residential buildings worldwide.
KNX has been approved as a European Standard (CENELEC EN 50090 and CEN EN 13321-1), an International Standard (ISO/IEC 14543-3), a Chinese Standard (GB/Z 20965) and a US Standard (ANSI/ASHRAE 135). It groups about 7000 KNX certified products from 200 member companies, installed by more than 30,000 KNX partner installers in 100 countries.
KNX is designed to automate and control lights, shutters and heating systems in homes and buildings.

LonMark Intl. Echelon Corporation [13] is an international company that targets a worldwide transformation of the electricity grid into a smart grid. To achieve this task, Echelon developed LonWorks, a family of products able to interact with each other using the LonTalk protocol. The promotion of LonWorks products to end-users, manufacturers and integrators is ensured by the LonMark Intl. [14] organization. This organization is also responsible for giving guidelines and help to manufacturers, integrators and end-

10. http:grouper.ieee.org/groups/1901/2/
11. http://webstore.iec.ch/preview/info_isoiec18000-7%7Bed3.0%7Den.pdf
12. http://www.knx.org
13. http://www.echelon.com
14. http://www.lonmark.org

users to build or simply use LonMark certified products. Lastly, its role is to ensure the interoperability of all products, by verifying that each of them meets the LonMark guidelines to operate on a LonWorks network. The LonTalk protocol designed by Echelon is currently recognized as an international standard (ISO/IEC 14908), a European standard (EN14908) and a Chinese standard (GB/Z 20177.1-2006). In February 2009, over 700 organizations joined LonMark.
LonWorks products are mainly used for technical building management concerning lights and HVAC (Heating, Ventilation and Air Conditioning).

BACnet Developed under the auspices of the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), BACnet [15] is a Data Communication Protocol for Building Automation and Control Networks. It has been released as an American national standard, a European standard, a national standard in more than 30 countries and an ISO global standard. The protocol is supported and maintained by ASHRAE Standing Standard Project Committee 135, divided into 13 working groups. These groups work to integrate issues from various building aspects from "Objects and Services" cooperation to elevator or lighting management. In February 25, 2011, 503 Vendor IDs were issued from all over the world.

3.1.4.3 Application Protocols

Universal Plug & Play(UPnP)
Universal Plug & Play (UPnP) [16] is an application level protocol which aims to ease the connection and use of electronic devices, based on a discovery-search mechanism. As a UPnP-Device joins the UPnP network, it sends an XML description file to all UPnP Control Points. This description provides other devices on the UPnP network with information such as manufacturer, device type, device model or the unique identifier of the device (UUID). The services offered by a UPnP device are also specified in its description. Each service conforms to a type, a set of UPnP actions that the service renders. Providing such information to other devices on the network makes it possible to use the functionalities of the device without having to install anything (thanks to the standardization of the descriptions and method call mechanisms).
UPnP is particularly used for multimedia management. The Digital Living Network Alliance(DLNA) extended the UPnP-AV specifications, to provide commonalities in media descriptions, format negotiation and connections between media servers and renderers.

Device Profile for Web Services (DPWS)
DPWS [JMS05] could be considered as a next generation UPnP. If the goals are the same, DPWS uses WebServices as a transportation mechanism whereas UPnP uses simple XML over IP. DPWS is still based on the concept of services, devices, operations and parameters. It includes numerous extension points, allowing for seamless integra-

15. http://www.bacnet.org
16. http://www.upnp.org

tion of device-provided services in enterprise-wide application scenarios.

DNS-SD [17] is a service advertisement and discovery mechanism that relies on the DNS standard. It uses DNS packet formats, servers and programming interfaces for browsing for services. DNS-SD is compatible but not dependent on Multicast DNS(mDNS). This standard is used, for instance, by the Apple Bonjour protocol for locating printers, other computers or different services such as media sharing. Some projects such as iDo [18] uses this protocol in a home automation context.

SIP

Session Initiation Protocol (SIP) is a protocol for initiating, modifying, and terminating an interactive user session that involves multimedia elements such as video, voice, instant messaging, online games and virtual reality. Developed by the IETF MUSIC Working Group, it was initially published in 1996 as RFC 2543 and updated by the RFC 3261 in 2002. SIP aims to ease the establishment of communications between multimedia devices using two protocols: RTP/RTCP and SDP. When RTP is used to transport voice data in real time (the same as H.323 protocol), the SDP protocol is used to negotiate the participant capabilities, codification type, etc. SIP has been designed in conformance with the Internet model and all the logic is stored in end devices (except the routing of SIP messages).
SIP can establish sessions for features such as audio/videoconferencing, interactive gaming, or home appliances over IP networks. It is based on requesting and answering messages and can reuse many concepts from previous standards such as HTTP and SMTP.

XMPP

The Extensible Messaging and Presence Protocol (XMPP) [19] is a protocol for real-time communication such as instant messaging, voice and video calls, collaboration or lightweight middleware communications. The core technology of XMPP was invented by Jeremie Miller in 1998 and formalized by the IETF in 2002 and 2003 in RFCs. The latest RFC reviews for XMPP are RFC 6120, 6121 and 6122 published in 2011. The XMPP community continues to define various XMPP extensions through an open standards process run by the XMPP Standards Foundation. An active community of open-source and commercial developers produces a wide variety of XMPP-based software.

The complexity of the home automation domain clearly appears from this presentation. Taken individually, each product, transportation protocol, or medium is quite easy to handle. The complexity is due to the huge number of possible solutions for a given problem and to the difficulty of becoming aware of all the benefits and drawbacks offered by each solution.
Home automation can be used in various contexts, such as assisted living or energy

17. http://www.dns-sd.org/
18. http://hw-server.com/ido-smart-automation-everyone
19. http://xmpp.org/ (May 2011)

saving. Professionals in these domains are not aware of the possibilities offered by home automation technologies and home automation engineers are not familiar with each domain's problems.
Tools are required to simplify variability management and conciliate home automation technologies with domain-specific problems to bring new solutions.

3.2 General purpose approaches

The ageing of the European population and the complexity of home automation technologies require a software tool, able to ease the creation of user-specific, installation-specific software, using home automation devices to provide a personalized service.
The development of this tool should rely on an approach adapted to the problem we are trying to resolve here. This section surveys several general purpose approaches and tools, to evaluate their position in relation to our requirements.

The survey starts with an overview of Internet of * concepts together with service-oriented architectures. Afterwards, this section presents tools and approaches from the component-base engineering field. This presentation ends with approaches that conciliate services and components; namely component-base architectures for service-oriented applications. Each of these three parts is organized the same way: it firstly describes the general idea of the approach and then details some tools implementing the approach.

3.2.1 (Web)Service-Oriented Architectures

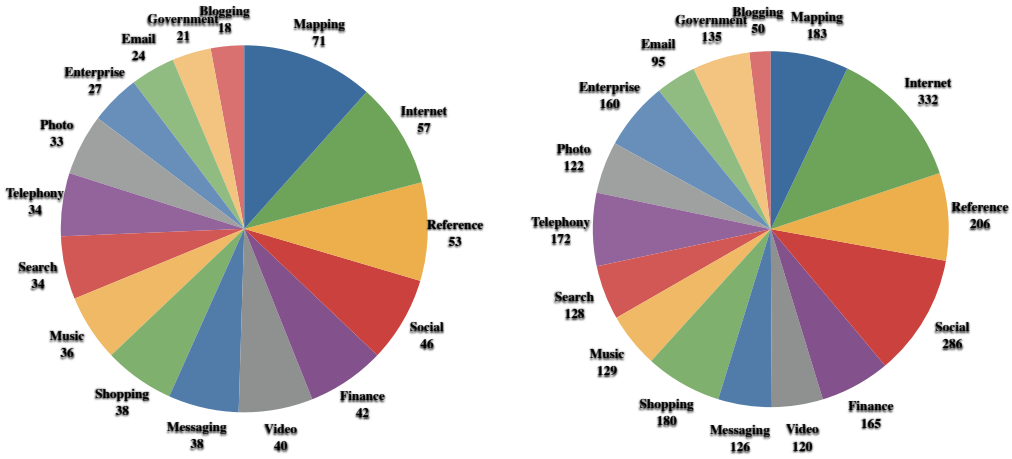
3.2.1.1 Internet Of * and the Cloud

The Internet we know aims at sharing information and facilitating interactions and communications between people or computers.

In an article [DFD+09] extracted from the book "Towards the future of Internet", Domingue et al. reports a chart from the web site ProgrammableWeb.com, presenting the number of available APIs (services offered through the web) in fifteen categories. These numbers extracted in November 2008 are reproduced in the chart presented in figure 3.1a. Just next to this figure, another chart (fig 3.1b) presents the current numbers from the same web site, for the same categories, but three years later. If the repartition has not significantly evolved, the total number of available APIs considerably grown form 612 in 2008 to 2589 in 2011 (plus 423%).
With the evolution of Internet technologies, the services offered through Internet are getting more and more numerous and rich.

Internet Of Services
Synergy is probably the best word to describe the Internet Of Services (IoS) goal. Many services are now available on the Internet, such as hotel bookings, flight reservations or car rental. IoS aims to orchestrate the interactions of existing services to create new services more integrated, and dedicated to a task or a user. The goal is to create, for instance, a "Travels booking service" which aggregates the booking of a flight, hotel and car in a unique process.

Among all the "things" able to provide services, everyday life objects are also getting enriched with abilities to communicate through Internet-based technologies.



(a) APIs, 2008 (b) APIs, 2011

Figure 3.1: Data from [DFD+09] and actualized from ProgrammableWeb.com

Internet Of Things

Stemming from the evolution of electronic devices, the Internet Of Things (IoT) relates to an approach in which objects of everyday life have reached a sufficient level of maturity to interact one with each other. This interaction gives them the ability to act differently according to the situation 'sensed' with a stronger added value. Still in its infancy, the IoT is looking for software tools to develop and describe these interactions, as much as the services rendered.
Software components are quite suitable for virtualizing everyday life objects, and service-oriented computing is suitable for describing and implementing interactions. Software solutions developed in the future would probably merge both of these approaches. As long as it can be considered that a component offers services to other components, the collaboration of services and components produces promising results.

Information, devices and services are now able to communicate through the Internet, and render more and more integrated services to users. This strong integration requires all these services to be able to self-locate their dependencies, with no need for the user to set up technical properties (i.e.: server IP address, tcp port, etc...).

The Cloud

Surfing on the service wave, hardware providers, software providers and others are offering more and more "things" as a service. Among others, Software as a Service (SaaS) and Platform as a Service (PaaS) are the two main paradigms showing that everything can now be provided as a service.
The Cloud could be defined as the Internet of tomorrow. As everything is being served

as a service, there is no more need to precisely locate services. Do you need a printer? Ask the cloud for the closest printer from you, and just use it. Might you need a lawyer? Ask the cloud for the best of them, and have a video-conference with him.
This approach makes it possible to obtain the hardware configuration you need, just-in-time, to run your software as a service. You will never know where your software is really run, but it will run in the cloud.

These principles are built on top of software architectures which basic elements are services or resources. Therefore, these approaches for building software are called Services-Oriented and Resource-Oriented architectures.

3.2.1.2 Architectural principles

In [Pap03], M. Papazoglou defines Service-Oriented Architecture as "*a way of re-organizing software applications and infrastructure into a set of interacting services*". Services are able to describe themselves and realize computations for applications or other services, from a simple method execution to a complex business process. Services are offered by service providers which are responsible for the supply and support of their services. Clients of these services can be either applications or individuals, internally to the organization or from outside.
Possibly distributed over Internet, services must be *Technology Neutral* to be provided and used whatever the context, *Loosely coupled* to not rely on any specific context of use, and support *Location Transparency* by registering their location and description in directories such as UDDI, to allow clients to call the service regardless of its location.

SOA vs. WSOA
In the Software Services community, service-oriented architecture is often used instead of web services-oriented architecture. However, there is a clear separation to respect between service-oriented architectures and web service.
Then, if a service is accessible through the Internet, this service is called web-service, and is a particular case of a service. Then service-oriented software can use this service as it would any other.
In conclusion, a service-based application does not necessarily use web-services and may even use no web service at all; there are other means of creating software based on services.

Web-Services
A Web Service renders a service, using the Internet as a support. Web services are composed of methods that can be called by clients. Customers do not have to care about how the service is given and can just use it.

The use of Web Services is based on a "search and use" mechanism. Service providers are responsible for the registration of their services into a Universal Description Discovery and Integration (UDDI) directory. When a client wants to use a service, he

first searches in a service directory. Each registered service comes with a description, which helps clients in their service selection process. The real call to the service is made directly from the client to the server. Figure 3.2 illustrates this mechanism.

Registrations and descriptions of services in the UDDI are based on a description language for web services called WebService Description Language (WSDL). The description of a service informs users about the list of operations offered, parameters, and types of objects manipulated. Dynamic discovery and use of services are enabled this way.
Communications between clients, servers and UDDI instances use a unique carriage protocol called SOAP. SOAP has been based on XML descriptions in order to make use of HTTP, SMTP, and other application protocols as carriers.

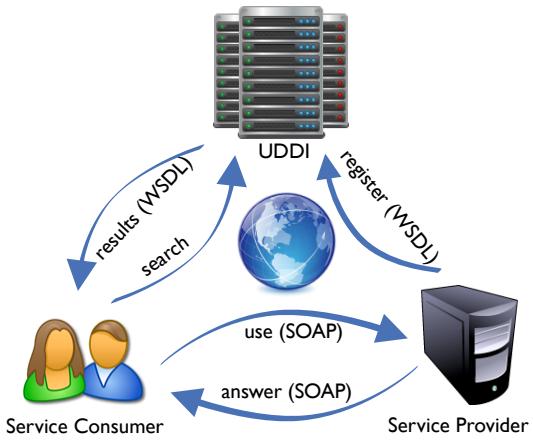


Figure 3.2: WebService Architecture

Although this approach provides mechanisms for dynamic search and use with precise descriptions, the amount of data exchanged and the complexity of the SOAP message structure can become an issue. Resource-limited platforms may not be able to embed a SOAP parser, or have a power supply designed to send a large amount of communication data. To cope with this problem, Resource-Oriented Architectures have been proposed.

Resources-Oriented Architectures

In his doctoral dissertation in 2000 [Fie00], Roy Fielding introduced the term and idea of Representational State Transfer (REST).
REST has been designed to lighten transfers of information through web-based communications. Instead of heavy serializations of concrete program objects, REST architectures are share representations of these objects using XML for instance. These representations handle all coherent and meaningful information for the request (resp. answer) to be processed by the server (or client). REST defines only four standard CRUD [YWDJ98] operations to manage resources.
GET is to retrieve the resource pointed by the called url. **POST** asks the server to add the information contained in the request (hence a resource representation) to the resources pointed at the requested url. The **PUT** operation is used to create or entirely replace a resource, based on the representation contained in the request. Finally, **DELETE** requests the server to delete the specified resource.
URLs of REST servers handle information about the resource concerned by a request.

For instance, a GET request on a URL such as *http://myMedia.org/* would be answered with an XML representation of the entire media library; a POST containing information about a new book, called on the *http://myMedia.org/books* would result in the addition of this book in the book library. As a final example, a DELETE request on *http://myMedia.org/books/2517* should remove the book whose unique identifier is 2517 from the book collection.
For its transportation, REST was initially described in the context of HTTP but is not limited to that protocol. Supported by an application-level transfer protocol, REST is thus development-technology agnostic.

Thanks to the appearance of these two paradigms, ideas emerged about connecting software systems and everyday life objects. The presence of registries or auto-discovery mechanisms also led to an abstraction of the details of the physical location. These interconnections finally brought new paradigms known as the Internet of *.

3.2.1.3 OSGi

OSGi [All11] is an association created in 1999 aimed to provide facilities for software integration and development. To achieve this task, the association released a set of specifications defining what any runtime implementation should do to reach a given service level. These specify which minimum set of services a runtime must offer to be compliant with the level.
In OSGi, services are given and contained in deployments units called *bundles*. Each bundle contains a *Manifest* file giving information about the runtime dependencies, the classes offered and some other information about what the bundle needs to run. Bundles can be installed, updated or removed at any time and their services can be started and stopped, with no need to restart the runtime platform.
Services are defined by Java interfaces (for Java runtime implementations), and are stored in the runtime context. Thus, any client on the platform who needs a service can search in the context registry for the service they need. Service method calls are locally handled inside a JVM, which makes this service-oriented runtime much faster than web service-based applications.

In OSGi, relations between bundles are never made explicit. Worse still, relations between bundles can be due to service dependencies that are just hard-coded and can only be changed by rebuilding and redeploying the bundle. In a static application, with few updates in time, this is not a real problem. Moreover, there are very few reflection primitives and thus, interactions between bundles can hardly be traced or even made explicit. OSGi do not address issues of interoperability, variability management nor safety and security. The registry mechanism is a good point concerning openness, and adaptations and evolutions of applications are simplified by the native lifecycle management of OSGi. In a word, OSGi lacks of a clear means of visualization of the architecture while running, but makes a very good base.

The table below is completed for each approach or tool, and presents strengths and weaknesses in a synthetic way. Individual tables are merged altogether in section 4 as a synthesis.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
	+	+	+		

3.2.1.4 Enterprise Service Bus

Enterprise Service Bus (ESB) refers to a business middleware family for service-oriented applications. This middleware acts as the only mediator of services in the enterprise, by providing a runtime environment for deploying business services. Legacy software can be integrated as services into the business service orchestration. These are declared as any other service, within the scope of the ESB runtime. The specifications have been successfully implemented in several frameworks such as OpenESB by Sun-Microsystems, ServiceMix by Apache Foundation, or Petals by OW2.
Java Business Integration (JBI) or JSR 208 is an industrial Java standard developed to ease the integration of software systems over Service-Oriented Architectures. It uses an ESB as a basis to define a component model. It has been designed to reuse Java technologies such as WebServices, BPEL or JMS, and thus avoid new specific developments. In JBI, components have an independent life-cycle and communicate through their services over normalized message middleware. In fact, this middleware acts as an abstraction layer for communications and eases the integration. JBI components are split into two categories:
Service Engine Components are directly hosted by the JBI runtime environment, and are in charge of message processing, routing or orchestration of services. They cannot communicate outside of this scope.
Binding Components expose or consume standard JBI services, and perform the bindings with external non-standard software.
The packaging as components is described in the framework like this: service descriptions are encapsulated into Service Units, which are then encapsulated into deployable business components called Service Assemblies.

The message middleware makes JBI a serious candidate in terms of interoperability of components. Openness is offered by the Binding Components, and evolutions are natively supported thanks to its service-oriented nature. Besides its good properties, there is no clear separation between types and instances of services/components. Moreover, no introspection of services is offered and interconnections between components are not explicitly expressed, and sometimes even hard-coded inside the components. This lack of clarity in the component interdependencies makes it impossible to dynamically replace components and/or reason about the system's state. This approach does not target the resolution of variability management or safety and security issues.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+	+		+		

3.2.2 Component models

3.2.2.1 Description

Douglas McIlroy first introduced the notion of software component in 1968 at the NATO conference. This new paradigm defends a mass reuse of existing components and the creation of software as assemblies of components.
As described in [MT00], component models are made of three essential elements. A *Component* represents a computational unit and can realize an entire application or a single method. Components have a type and expose interfaces to collaborate with other components. This collaboration is enabled by component *Connectors*, in charge of the communication between the application's components. These connectors are typed and can have different behaviors, roles in the application, and can make use of various techniques to support components' interactions. Finally, a *Configuration* describes a particular assembly of components and connectors, and specifies the component-base software system.
The next section provides a brief list of component models.

3.2.2.2 Darwin

In [GMK02] Ioannis Georgiadis and al. present a component model to describe self-organizing software architectures of distributed systems. In this model, components are defined by component types and can have multiple runtime instances. Instances can be statically specified at design time, or created on demand at runtime. Usually, components provide and require services. The provision or requirement is made through typed ports. These types are specified by the interface of the service they offer. Components are connected by their ports; ports that have to be of the same kind. The semantics of bindings is a classical service call. Components can be assembled into composite components. Their specifications describe the instances used and how they are connected. At runtime, component instances embed a view of the global configuration and a manager handling the architecture constraints, in charge of maintaining the configuration view synchronized with the system state.

The clear separation of types and instances is a plus. Runtime creation of instances can help in adapting the runtime to the environment. The configuration view synchronized with the runtime is a very interesting property. The use of Java class loading to change utility functions used by the policy manager is a good way to runtime evolution. If the typing of ports is helpful to guarantee the consistency of an application model, the typing at the implementation level can act against the interoperability property, but

enforce the safety of the system. Bindings have a clear semantic, but cannot use other communication links than the one they have been designed to use (Java RMI in this case). It is a limitation. The adaptation policies (in case of binding loss or component arrival for instance) are hard coded and distributed in each instance. Thus they cannot be easily changed at runtime.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
		+	+		+

3.2.2.3 Koala

The Koala component model has been designed to handle the increasing diversity and complexity of embedded software and decrease development costs. Rob van Ommering and al. explain in [RvdLKM00] that a way to achieve this is to model the software architecture and reuse existing software components rather than re-implementing the wheel. In their approach, a clear separation is made between component development and system configuration. It means that component developers cannot make any assumption about the context of use of each component and designers cannot change component behavior.
Components can require services provided by other component's ports. A configuration describes an assembly of components. It handles the model of the application. To help in describing system assembly, Koala offers compound components in which instances to be deployed and their interactions (bindings) are described. In this case, an action on a port of the compound component may have to be forwarded to internal components (e.g.: for initialization). To eliminate the ordering problem, Koala introduced *Modules* to handle one-to-many, many-to-one or many-to-many bindings. They are in charge of propagation and have a pre-defined treatment.

The clear separation between component development and assembly creation (for a particular application) is the key to success. Composition mechanisms are also welcome to cope with diversity management and promote the reuse of existing components to create value-added compound components. Here again, connection ports are typed and should conform to a specific interface. This is an advantage for securing the software but may cause problems in future evolutions, in terms of interoperability. Koala introduced modules to handle specific communications between components and act like a proxy. They could have gone a bit further by systematically using modules to specify each connection. This information could help in solving issues, or in specifying the system more precisely.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
			+	+	+

3.2.2.4 Fractal

Fractal [BCL+06] is a modular and extensible component model to design, implement, deploy and reconfigure various systems and applications. Famous implementations of Fractal are Julia and AOKell (Java), Cecilia (C), FractNet (.NET) and FracTalk (SmallTalk).
The Fractal component model supports the definition of primitive and composite components. Each Fractal component consists of two parts: a controller, which exposes the component's interfaces, and content, which can be either a user class or other components in composite components. The model makes the bindings between the interfaces provided or required by these components explicit, as well as the hierarchic composition (including sharing).
Primitive components contain the actual code and composite components are only used as a mechanism to deal with a group of components as a whole, while potentially hiding some of the features of the subcomponents. Primitives are actually standard Java classes (in the Java distributions of Fractal) conforming to some coding conventions. Fractal does not impose any limit on the levels of composition, hence its name.
All interactions between components pass through their controller. The model thus provides two mechanisms to define the architecture of an application: bindings between interfaces of components, and encapsulation of a group of components into a composite. By default, Fractal proposes 6 controllers that may be present in components: Attribute, Name, Binding, Content, Lifecycle and Super Controller.
DigiHome[RHT+10] is a communication middleware built with Fractal. Its main objective is to offer a support for REST communications, and complex event processing, in a context of home automation.

Reflective execution platforms like Fractal or OpenCOM [BCU+04] do not provide a clear distinction between the reflection model and the reality. Modifying the reflection model implies a modification in the runtime. If this offers a means for adapting the runtime, there is no means to anticipate the effect of a reconfiguration, before actually executing it. Nor is there means to execute what-if scenarios to evaluate different possible configurations, etc. This point is a drawback from a safety viewpoint. This lack of an explicit and independent reflection model imposes that most of the verifications must be carried out while reconfiguring. Pre-condition on reconfiguration actions, as proposed by Léger [LLC07], are checked and roll-backs are performed if something goes wrong. In addition, component models such as Fractal are slightly opaque with respect to the outside world, making openness and reuse by third party applications complicated if not anticipated in advance. Lastly, the dynamicity of an application running over Fractal is compromised, because the deployment of new component' types can not be carried out without restarting.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
		+			

3.2.3 Component Models for SOA

3.2.3.1 Description

By nature, Service-Oriented software is dynamic and its architecture is not always easy to figure out. Indeed, the services used and the connections between software elements are never known prior to the execution because of the late binding principle. The late binding principle relates to the fact that a service to be used is searched and selected just before its call. Component models for Service Oriented Application / Architecture (SOA) have been invented to try to make the description of this kind of application more explicit. They merge well-known software component techniques with new services ideas, and tend to conciliate the best of each approach.

Components, as defined by component models, provide services to other components through their ports. Components' ports are defined by an API. Services, from service-oriented architectures are intended to be used in orchestrations to create value-added applications.
Since both of these concepts offer services, new component models have been designed to merge both paradigms. This section presents some famous implementations of these component models.

3.2.3.2 SCA

SCA [20] provides a component model for both the composition of services and for the creation of service components. SCA is a model that aims to encompass different programming languages, frameworks and environments commonly used to build components and services, such as Web Services, Messaging systems and Remote Procedure Call (RPC) for communication purposes. Its goal is to setup a single and common way to access and assemble service-based applications.
SCA can be presented through four major parts of the specification.

Specifications

Assembly defines how components are packaged as services and how they can be combined into composites that perform a particular task. Composite components can be used as classical service components, which simplifies their reuse. Assembly in SCA also defines how components and composites are connected. Functional service properties, such as data encryption, or authentication, are described outside the service business code, which saves developers' valuable time. Indeed, it enables the modification of the connections or the properties, without changing the business code.

Client and Implementation Model defines how services are packaged and accessed in various languages. API implementations in Java, BPEL, C++, Javascript or the Spring Framework, offer means to package a service or access any SCA service. For

20. http://www.osoa.org/

development concerns, it means that there is only one interface and packaging method to learn to provide and use any SCA service. This interface makes it possible to access to the component using Web services, JMS, JCA and EJBs natively. Here again, service properties are described outside the code, to make changes much easier.

Policy Framework is aimed at offering means for the definition of security, authentication, quality of service, and other important policies of a service. In fact, the SCA Policy Framework makes use of the WS-Policy and WS-Policy Framework open standards, as a support to describe policies. This way, descriptions of policies such as "any data sent to this service must be encrypted" or "the user of this service must be authenticated" are made available. Here again, policies can be defined outside the business code of the service.

Bindings specify the mechanisms that can be used to access or connect a component. Bindings can be implemented using Web services, JMS, JCA, EJBs or any other communication way. Keeping the consistency of the approach, bindings are defined outside the component business code.

SCA is a standard, a set of definitions describing how such a system should behave. Thus, it imposes certain types of implementation, in order to guarantee the presence of mechanisms for openness, evolution or remote control.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
	+		+		+

3.2.3.3 FraSCAti

FraSCAti [MRRS10] is an implementation of the SCA specifications. It is certainly the closest approach to what is required. Components can be composed into composite components. Communications between components are made using services and several communication media can be used. These elements make FraSCAti a serious candidate to address openness and remote control concerns.
In the past months, efforts have been spent on integrating FraSCAti and OSGi, which have improved its faculties of evolution. In terms of interoperability, FraSCAti offers mechanisms for the connection of services, but does not directly address the integration of smart devices. Thus, interoperability of components in our context is still compromised by the use of APIs, for the definitions of services rendered and required, by ports. If two components have not been designed to be connected, an ad-hoc connector has to be created.
The FraSCAti script tool enables reconfiguration and adaptation of component assemblies. But adaptations are limited to the manipulation of binding and component instances, whose types are available on the platform. New instances can be created, new bindings can be set, but no new types can be installed using FraSCAti Script.
The variability of FraSCAti itself is managed using a Software Product Line(SPL). Each runtime instance of FraSCAti is a product of the SPL. Thus, features of FraSCAti can

be deployed on demand. However, this variability management concerns only the internal features of FraSCAti, and an external tool is still required to handle the variability of applications made with FraSCAti.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+	+		+		+

3.2.3.4 iPOJO

iPOJO [EHL07] is the Apache service-oriented component runtime built on top of OSGi SOA platforms. iPOJO is the natural evolution of the Service Binder mechanism introduced by Cervantes et al. in [CH04]. The iPOJO framework merges the advantages of component- and service-oriented paradigms. Specifically, application functionalities are implemented following the component paradigm. Each component is fully encapsulated, self-sufficient, and provides server and client interfaces as services. An iPOJO component is actually managed by a reusable container, which provides common middleware functionalities. Each component container can be configured with a different set of middleware services.
The iPOJO component model focus on providing management facilities for pervasive applications based on component model, and a service-oriented runtime. This general purpose component model for service oriented architectures provides a solid base for domain specific extensions and developments.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
	+	+	+		

3.3 Domain-specific approaches

General purpose approaches make it possible to design and implement software solutions for many problems. But this flexibility comes with a complexity due to the knowledge required to be able to develop or just use these approaches.

Domain-specific approaches tend to reduce this complexity to a minimum, by providing tools on top of general purpose approaches adapted to a specific domain of use. In this section, several tools built upon this principle are reviewed to list their advantages and disadvantages.

3.3.1 Description

Model Driven Engineering (MDE) is an approach that promotes the use of an abstract representation of a piece of software, before its actual realization. From this abstract view, tools and methods make it possible to automate the final software generation, tests and validations across pre-defined requirements. Models are human understandable representations of reality. They can handle information about structure, data exchange, communication links, or some building constraints of a piece of software.
Domain-Specific Language (DSL) are another means of abstraction and description of software systems. Dedicated to a specific domain, they can be graphical, textual or both. They are designed to restrict the concepts manipulated to the ones from the application domain. This approach makes it easier for domain specialists to express a software system architecture and behavior, by using their own terminology.

The goal of these tools is to provide a sufficient level of abstraction to make software development easier, more flexible, with an enhanced level of reliability, and shorter time-to-market.
The rest of this section presents several approaches built around the concepts of MDE and/or DSL, that simplify the creation of applications in the domain of home automation and/or pervasive computing.

3.3.2 Projects

3.3.2.1 uMiddle

Nakazawa et al propose a framework that bridges remote smart spaces called D-uMiddle in [NT07]. This makes it possible for a device to interact with another, over the Internet. This is made available by four distinct features of D-uMiddle. Firstly, a local mapper mechanism abstracts sensor nodes into common representations. Secondly, a mechanism translates data transmission protocols from a node-specific one to a D-uMiddle common one. Thirdly, a remote mapper mechanism creates proxies of sensor nodes from remote smart spaces in the local space. Fourthly, a transport module enables devices to receive data over IPv4 NATs network. The consumer devices, as a

result, can use sensor nodes in remote smart spaces without depending on their own protocols and semantics, and without being burdened by the complicated IPv4 NATs.

D-uMiddle brings a solution for connecting remote smart spaces, with no need for a developer to care about transportation. Remote control of equipment is thus made possible for free. Nevertheless, it does not supply tools to handle variability, adaptation or evolution of a deployed system, and nothing is specified about the securization of the connection.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security

3.3.2.2 SOPRANO

SOPRANO (Service-Oriented Programmable Smart Environments for Older Europeans) was an Integrated European Project, which successfully ended in April 2010. Its main achievement was the release of openAAL [WSO+10], a framework built on top of an OSGi execution platform. OpenAAL helps in getting information from devices, and acting on them from a higher level of abstraction. Its framework integrated a context manager, able to give a virtual view of all devices, a process manager in charge of making decisions for any change in the context, and a composer, dealing with the actual services for interaction with the real environment.

OpenAAL proposes a solution to efficiently built applications ready to evolve with needs and able to adapt to changes in context. However, no attention is paid to the variability management, remote control or interoperability of devices. Nothing is said about safety of security.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
		+	+		

3.3.2.3 Gaïa Framework

Gaïa [RHC+02] is presented as a meta-operating system for ubiquitous computing, built on top of a classical operating system. Its goal is to detach itself from the heterogeneity and complexity associated to ubiquitous environments. Gaïa is composed of a Kernel, responsible for the runtime management of applications and a Framework to build these applications. An application runs in an Active Space, a physically-limited space where services and devices are available for ubiquitous computing.
Each Gaïa instance is specifically configured for the active space it manages. To allow for describing Gaïa applications for several active spaces, Olympus[RCAM+05] proposes a high-level DSL working with virtual entities. From an Olympus application model,

the underlying Gaïa OS takes responsibility for mapping each virtual entity to a service, or device, available in the active space.
It has been implemented in CORBA and can be ported to other communication middleware architectures such as SOAP or RMI.

The interoperability of services and devices is ensured by the common set of basic services. Adaptations and evolutions are made possible by the Component Management Core of Gaïa which can dynamically load, transfer, create or destroy components or applications. Remote control is made available by the underlying CORBA platform, in the implementation described. Variability management of components or applications, safety and security, and openness of the solution are not targeted by this work.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+		+	+		

3.3.2.4 DiaSuite

DiaSuite [21] is a software tool suite designed to ease the creation of pervasive and/or distributed applications. DiaSuite [CBC10] is composed of several elements. DiaSpec is the Architecture Description Language (ADL) of the suite, used to describe the applications at a convenient level of abstraction. From this description, DiaGen automates the code generation of the application, and DiaSim provides the support for the test, simulation and validation of the generated application. As an example, Bertrand et al. present in [BCJ+10] how they used the SIP protocol as a generic communication bus for a pervasive application developed with DiaSuite tools.
This tool suite has been augmented with Pantagruel [22], a visual DSL created to support the development of pervasive applications. A first step when using Pantagruel aims at defining the entities involved in the future application domain. In a second step, entities of the application are orchestrated in order to define the logic of the application. A last step generates an application code, compatible with the DiaSuite tools. Details about this tool are available in [DMC09].

These tools meet the demands of a tool chain to develop pervasive applications from a high-level description. Code generation and the simulation environment are very good tools to improve the efficiency of the development process and the reliability of the code, as much as facing the variability of solutions. Designed to ease the development of pervasive applications, these tools do not address issues about variability management, application evolutions or adaptations.

21. http://phoenix.inria.fr/projects/diasuite
22. https://pantagruel.bordeaux.inria.fr/

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+	+			+	+

3.3.2.5 Habitation

Habitation is a methodology, a set of tools to address the specific requirements of home automation application development and design. In [JRS+09], Jimenez et al. describe how the combination of a DSL, and an MDE approach eases the creation of solutions in this domain. Habitation proposes three main tools. A catalog of functional units centralizes elements that can be reused in various applications. Home automation devices are composed of several functional units. The second tool is a workspace in which elements of the catalog can be placed to define a specific application. Called the application view, this tool aims to provide tools for the assembly work and make it accessible for non-domain experts. The last tool is a kind of engine, which translates from the model and DSL description, to a technology-specific configuration file.

The approach proposed by Habitation is very promising and sounds helpful in providing non-expert users with tools having a sufficient level of abstraction to be user friendly. However, Habitation only addresses pre-deployment design issues and does not deal with issues such as evolutions, adaptations and safety and security issues. Such as DiaSuite, Habitation remains an good approach to face the variability issue, while improving the efficiency of the development process.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+				+	

3.3.2.6 Wired Application Description Language

Wired Application Description Language(WADL) is a language designed to ease the description of dynamic applications, and provide an explicit view of the relations between elements. In [CDT08] authors present how WADL has been used in the creation of a dynamic sensor-based application.
WADL has been implemented on OSGi(see section 3.2.1.3) and relies on the WireAdmin service offered by the execution platform. In this implementation, wires between producers of information and consumers are dynamically created or deleted, according to the elements available in the system. Wires are specified by two filters. Each filter is used to make a selection among all available services, and capture producers' (or consumers') services required for the wire.

WADL provides a tool to explicit the architecture of dynamic applications, which is often difficult to extract because of the runtime evolutions. By nature, this approach copes with the adaptation requirement. The interoperability is realized by the use of a

Producer/Consumer pattern. Evolution is supported by the filters that can be flexible enough to admit future evolutions. Issues on openness, variability management and safety and security are not treated in this approach.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+		+	+		

3.3.2.7 PervML

Muños et al. present PervML in [MPC06, MSCP06] in the context of the management of a pervasive meeting room. PervML is a Model Driven Approach designed to ease the development of pervasive systems. This language separates the analyst's view, describing the requirements of the system at a high level of abstraction, from the architect's view, where devices and implementation details are specified.
This abstract model of the system is then used in a tool chain which ends up with an executable OSGi(see section 3.2.1.3) code. This tool chain, detailed in [MP06], firstly transforms the platform independent PervML model to an OSGi dependent model, and then generates the executable Java code.
PervML and the associated generation tool chain are available as a plugin for Eclipse [CSMP07].
In [SVP10], authors explain how they introduced system evolution capabilities to adapt the generated systems to changes in user behavior. Their solution uses a context model, to detect specific situations, and a task model describing the jobs to be executed for each detected context.

PervML offers a suitable solution. Developed to be executed on an OSGi platform, it offers adaptation, evolution, openness and interoperability mechanisms. As presented in [CGFP09], PervML also targets the variability management issue. So far, a drawback of this approach is that people have to be familiar with UML to model a pervasive application. Also, the use of a pre-defined set of service interfaces described in the framework may become a barrier for the flexibility of the solution. Safety and security issues are not addressed by this work.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+	+	+	+	+	

3.3.2.8 AutoHome

In [BDLM11], Bourcier et al. present AutoHome as an autonomic management framework for pervasive home applications. AutoHome is described as a middleware that extends the iPOJO component model, to create a framework to host autonomic home applications. Using this approach, authors aim to separate the design and devel-

opment of the application itself, from autonomic management components. They aim to enable the development of autonomic management functions, ease their integration with the applications, and finally, deploy the resulting autonomic application on execution environments shared with other applications. As a consequence, an application on top of AutoHome has the following architectural elements: middleware offering an autonomic service-oriented component and a context facility, a runtime that includes monitoring and reconfiguration abilities, a set of service-oriented applications which represent pervasive components to be autonomously managed, and a set of managers organized in a hierarchy.

AutoHome makes it possible to include, within the application itself, specialized components that monitor and react on component or platform events. However, this solution does not seem to offer means for variability management, or interoperability between components, and does not consider safety or security aspects.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
	+	+	+		

3.3.2.9 WComp

WComp[FHL+11, TLR+09] is a component model designed to support ubiquitous computing. It tries to address issues introduced by the mobility of devices, their heterogeneity and the dynamicity of the execution environment in this domain. To cope with these problems, WComp federates three paradigms. Event-Based Web Services, firstly, that plays an important role in facing the heterogeneity of devices, as much as the dynamicity, the extensibility and scalability of a ubiquitous system. Secondly, a Lightweight component-model (called SLCA) is introduced and used to compose event-based Web services, and expose a new service. The third paradigm used concerns the adaptation and makes use of the Aspect of Assembly concept. These aspects define adaptations on assemblies using structural descriptions. Adaptation are thus not specific to particular service assemblies and can be composed.

WComp offers several mechanism for adaptations, evolutions and interoperability, and does not seem to address openness, variability management and safety or security concerns. Finally, it looks like the type system of the components rely on the implementation language type system, which can limit the reuse and interoperability.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+		+	+		

3.3.2.10 Niagara

NiagaraAX is a software framework and a development environment that leverage the accessibility of a device toward an Internet access. The normalization proposed by NiagaraAX of the behavior and data gathered from several devices, enables the implementation of seamless, Internet-connected, web-based systems, whatever their manufacturer or communication protocol. This normalization has been enabled by the Niagara's unique, patented component model that transforms the data from diverse external systems into uniform software components. These components share the foundation for building applications to manage and control the devices.

NiagaraAX eases the creation of applications integrating services and devices, by providing a framework for the development of drivers and applications. Built as a service-oriented architecture, this framework promotes the interoperability of devices and openness of solutions through Internet-based communications. NiagaraAX does not target issues about adaptations of the software or its evolution at runtime. Safety and security are not part of this work.

Interoperability	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
+	+				

Chapter 4

Synthesis

Scientific literature abounds with proposals using different approaches to cope with interoperability, adaptation or remote control concerns in several applications.
Generally, service-based propositions sound helpful in targeting the interoperability of devices, but clearly lack description of the running application once deployed. They bring essential ideas to properly handle the arrival and departures of elements, since a service can be started and stopped at any time.
Component-based architectures provide an ideal abstraction level that meets the requirements for a virtual representation of home automation devices. However, the specialized interfaces used as descriptions for ports, may prevent the realization of unpredicted connections.
Using components for SOA is certainly the best approach for our concerns. Bridging components and services makes the benefits balance out the drawbacks of each.
Transversally, model-driven engineering methods and techniques come with a lot of tools for virtual element manipulations. They seem handy for runtime management of devices, for the description of software systems and for variability management.

Table 4.1 summarizes the advantages and the disadvantages of the approaches described in the State-of-the-art Review.

4.1 Good properties identified

All throughout existing approaches, some good design properties have been collected. These properties are not sufficient to address all our issues, but are still necessary to properly cope with challenges. Some of these properties had been suggested in [NBFJ09] to cope with the requirements.

Reflexive Model
Coming from the MDE domain, the goal is to obtain and keep synchronized, an explicit and independent model reflecting the architecture living at runtime. This model makes

		Interop.	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
Generic Approaches	OSGi [All11]		+	+	+		
	ESB [Cha04]	+	+		+		
	Darwin [GMK02]			+	+		+
	Koala [RvdLKM00]				+	+	+
	Fractal [BCL+06]			+			
	SCA [sca]		+		+		+
	FraSCAti [MRRS10]	+	+	+	+		+
	iPOJO [EHL07]		+	+	+		
Domain-Specific Approaches	uMiddle [NT07]						
	SOPRANO [WSO+10]			+	+		
	Gaïa [RHC+02]	+		+	+		
	Dia Suite [CBC10]	+	+			+	+
	Habitation [JRS+09]	+				+	
	WADL [CDT08]	+		+	+		
	PervML [MPC06]	+	+	+	+	+	
	AutoHome [BDLM11]		+	+	+		
	WComp [FHL+11]	+		+	+		
	Niagara [Tri08]	+	+				

Table 4.1: Summary of existing approaches

it possible to reason about the application state and perform any required operation with no risk for the running system due to decoupling. An adaptation engine, for instance, is thus able to select, test and validate an adaptation scenario on the model, before actually performing the adaptation on the running system [LLC07]. Component-based execution systems often offer introspection capabilities making it possible to build this kind of model.

Externalized coupling

For a system to be handled in the right way, interactions between its composing elements have to be explicit. Component Models and DSL offer a means of description for these interactions. A clear and explicit description of the relation between components gives a better understanding, and makes the analysis of the system much more accurate. It leads to better adaptation decisions, taking into account concurrency problems or dependency cycles for instance.

Moreover, this externalization and the description of interactions and dependencies enforce the independence of the elements composing the system. It also improves the flexibility of the system, thanks to the possibility of modifying of the resolution and connection policies with no need to deal with business components.

Hot deployment
The possibility for a service to be dynamically deployed or removed during the runtime of a system is an essential principle to be considered while dealing with flexibility, adaptations and evolutions. The execution platform must therefore, support dynamic deployments and adaptations of the application during runtime with no restart. This is a basic facility offered by SOA execution environments.

Loose coupling
Component Models promote the loose coupling principle, meaning that all components must have independent life cycles, and no execution dependencies with each other (in term of libraries). This is necessary to enable and ease the replacement of elements in a system. Indeed, inter-component dependencies may imply a huge alteration of the system to replace a single component, just because it depends on other components. It may also result in a more complex computational process of impacts for a change, or worse, an impossibility for the system to evolve or be adapted.

Openness
Interoperability and openness to third party applications/contributions are the reasons why service-oriented architectures have been designed. Their goal is to offer services in a standardized way, to allow them to be used by any other system: any third party application must be able to use the services offered. The Internet of Services makes use of interface descriptions and registries to expose services to the world.

4.2 Points of contribution

In the electronics domain, the number of components and their always-possible connectivity have offered technicians and engineers the means to create various solutions. Even many years after their assembly, electronic devices can still be repaired or completed with new features. The proposal made in this thesis is to take advantage of the electronic way of doing to improve the flexibility of software systems while keeping a high level of safety and security.
To this end, the contribution of this thesis can be described from three aspects :

- A new component model that improves the flexibility of software systems, by offering means of connecting any component to any other. This aspect addresses interoperability issues and evolution requirements.

- Modeling tools to create, modify and simulate component assemblies, check their consistency and validity before their (re-)deployment at runtime. Safety and security, as much as variability management are requirements covered by this aspect of the contribution.

- An execution environment built over a Service-Oriented runtime, to support the proposed component model, cope with adaptation requirements and evolutions at runtime, and validate the proposal

Part II

Thesis and Achievements

A Scout is never taken by surprise:
he knows exactly what to do when anything unexpected happens.
Sir Robert Baden-Powell

The study of the state-of-the-art in software engineering has highlighted the lack of a software solution to address all requirements identified in the context of home automation for Ambient Assisted Living. According to this observation, the goal of this thesis is to fill this gap by providing such a tool.

This section presents the achievements of this thesis. While the first chapter gives information about some central themes that drove this work, chapter 6 details the contribution. Then chapter 7 brings some information about the implementation and places the contribution in a classification framework.

Chapter 5

Contribution

5.1 Global ideas

All throughout the work on this thesis some recurrent ideas drove both the search for solutions and the development of the proof of concept.

5.1.1 Being inspired by electronics

Variability, interoperability and adaptation are qualifiers that appear regularly in electronics. Indeed, any integrated circuit has to be able to operate with any other. Signals exchanged between components may have to be adapted, in order for the signal to reach the shape required by the receiver. Various solutions using many different electronic components can be envisioned to fulfil a need. People in this domain had to find and deploy tools, such as data-sheets or simulators, to eliminate these constraints. This thesis was strongly inspired by this domain to come to a solution. The link between electronics solutions and the contribution of this thesis is stressed all throughout this section.

5.1.2 Making it possible

Scientific discoveries and advances are often due to hazardous reactions, unpredicted situations, and even due to errors. The software engineering tools of today limit runtime failures by cutting down the design elements to a set in which the interactions are well known. This mode of protection makes it difficult to design new systems, by using existing components in an unexpected way. Research or engineering phases must not be limited by these concerns.
This discussion can be compared with debates about static or dynamic typing in programming languages [Tra09, Tra10]: where static typing brings safety, it looses the flexibility of dynamic type systems.
This thesis work paid a strong effort in making type checking and validations policies highly flexible and customizable. It allows researchers and engineers to loosen checking rules, or deactivate validations to eliminate typing problems, and concentrate on

experimenting new behaviors.

5.1.3 Keeping end-users in mind

Products are too often released to be sold, without end-users tests. As a result, these products may be considered too expensive or useless. From beginning to end, the solutions offered by this thesis have been designed for targeted users. Tools and methods were adapted to reduce the gap between how people are intended to behave and the way they actually use the solution.
Two populations of users have been particularly considered. The first population is the community of engineers and technicians which requires some tools to ease their work. Secondly is the system user population, such as carers and elderly people, who just want to be able to interact with the system. In both cases, it calls for the tools and the system to be highly intuitive. Intuitiveness has been improved by presenting the system to elderly people and by using the tools to design solutions.

5.2 Overview of the contribution

The contribution of this thesis is threefold. (1) A new component model, (2) tools to handle models, and (3) a runtime environment. To go into detail, these elements are presented as interacting layers. Each of them targets a particular concern and their synergistic collaboration makes the solution. The different layers are visible in figure 5.1, which provides an overview of the contribution.

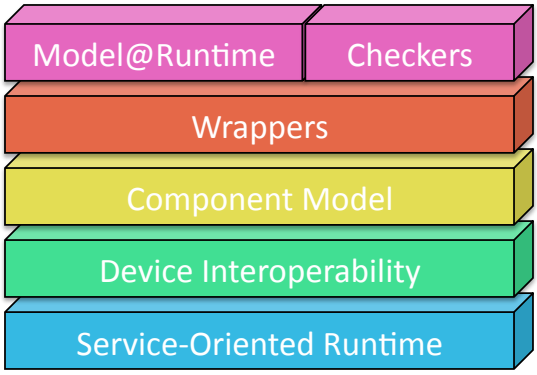


Figure 5.1: Overview of the EnTiMid layers

Device Interoperability addresses the mandatory need for *interoperability*. It is responsible for communication with real devices and their representatives in the *Component Model*.

The **Component Model** involves structures and methods, to handle abstract representations of real devices. It provides a unified description of available ports, param-

eters, and any other useful information for the *Model@Runtime* layer to work in good conditions. It enables the creation of tools to cope with variability, interoperability and safety concerns.

The **Model@Runtime & Checkers** layer concerns the necessary tools to ease the management of the system. The implementation specificities of components are invisible at this level, thanks to the *Component Model* layer. Simulations and checks can be safely performed at this level of abstraction, with no consequences on the running application. Model@Runtime enables the management of the system while running, and helps in dealing with variability management. Checkers offers tools for validation and improvements for the safety of the solution.

The **Wrappers** layer takes responsibility for publishing the devices present in the system, on application level networks. This ability opens our solution to existing and future, protocols and evolutions. Often too heavy to be embedded, this layer offers the devices, for free, an access through application level protocols.

The **Service Oriented Runtime** completes the contribution, by offering an execution environment for the new component model. It brings "life to the *Model@Runtime*" by providing the support for dynamic *adaptations* and *evolutions* while running.

Each level participates in meeting the requirements identified in chapter 2.3. Table 5.1 shows what concern is addressed by each layer. Separately, each layer does not satisfy all needs, but their collaboration does.

Takes part in	Interoperability	Openness	Adaptation	Evolution	Variability Management	Safety & Security
Model@Runtime			+	+	+	+
Wrappers		+		+		
Component Model	+		+	+		
Device Interop.	+					
Service-Oriented Runtime			+	+		

Table 5.1: Mapping layers to requirements

This contribution has been implemented. The runtime, called EnTiMid, has been developed on top of an OSGi platform. By the way, EnTiMid is a compound word from the Breton "En Ti", which means "In house", and "Mid", for Middleware. It is thus the middleware in the house. The component model has been developed using classical modeling techniques. Tools have been created to enable all functionalities.
Chapter 6 details each layer of this contribution.

Chapter 6

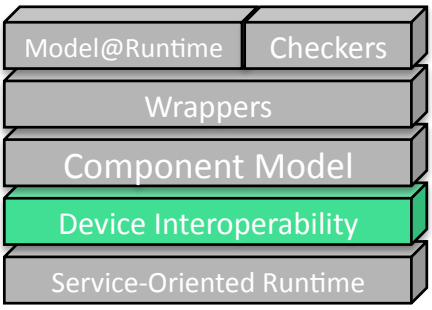
Details on strata

Like a geologist, this chapter dissects the contribution layer by layer. For each stratum composing the proposal, a section gives details on the roles taken on by the layer, its achievements, and its interactions with other strata.

The first section starts with the description of the Device Interoperability layer, which takes an essential role in the contribution since it enables heterogeneous connections. Section 6.2 then details the different concepts present in the component model, their interactions, and describes how the synchronization between component implementations and models is guaranteed. Section 6.3 explains how the Model@Runtime and Reasoning layer takes advantage of the component model to offer a great flexibility and multiple points for checking the conformance. The runtime environment chose is briefly presented is section 6.4, while section 6.5 introduces the Wrappers layer.

6.1 Device Interoperability

The Device Interoperability layer is probably the most important layer of the approach, since it answers the first requirement. Variability management, adaptations, or evolutions, would be compromised if only two devices were not able to communicate. Interoperability of devices is a central concern. It offers a foundation on which other layers can be built. This section presents how this interoperability has been realized.



In the domain of home automation, communication protocols used by manufacturers and their devices are not compatible. This incompatibility makes any direct interaction of devices coming from different brands impossible. To overcome this barrier, some manufacturers have worked in a consortium to define a unique communication protocol

for their respective products to be compatible. However, some of their products code Boolean values on a single bit, while others code it on a byte. Again, two products may not be operable with each other.
In [BRLM09] Bromberg et al. propose to automatically generate gateways between protocols, in order to address this issue. But building protocol-to-protocol translators solves the problem only partially, because the number of translators exponentially explodes with the number of protocols. Nevertheless, this proposition seems very interesting for an automatic generation of translators, from specific protocols to a higher-level of abstraction model.

6.1.1 Use of drivers

To realize this abstraction, drivers have been developed. A driver makes the link between real world devices and their virtual representation in a software system. Thus, they take on two responsibilities. In one way, they convert from vendor-specific communication messages to actions on their virtual representative. They also translate in the other way, actions on virtual elements into vendor-specific messages.
Secondly, drivers provide the virtual structures for each product they are able to interact with. All implementations specific to a given manufacturer are thus contained in drivers, or separate libraries. This makes the core system completely independent from devices' implementation specificities.

This independence implies the creation of a common structure, for the system to be able to properly handle devices in a good abstraction level.

6.1.2 Functional interfaces

This common structure may take the form of a set of programming interfaces. Each interface could specify a set of methods for a specific functionality. Then, drivers just have to provide objects, decorated with some interfaces, selected according to the abilities of each device. This set of programming interfaces has been created. A survey of devices functions has allowed us to extract the minimum set of common methods for each function as presented in figure 6.1 for instance. Once the set is defined, a library containing all interfaces was compiled and included in the framework.
The first experiments were promising. Interoperability was almost solved, but several drawbacks were rapidly identified while using this approach in real use cases. The set of interfaces is only extensible by augmentation of the framework. The development of a device driver could have fail because the required function interface was not available in the library. Moreover, direct method calls are not appropriate if the system has to consider a dynamic environment, in which objects unpredictably appear and disappear. In this case, object-oriented development using synchronous method calls becomes quite hazardous. Finally, if for any reason a component implementing the *LightControler* interface has to be plugged into a shutter, the operation is not feasible without an ad-hoc adapter (illustrated in figure 6.1). Interoperability was not solved.



Figure 6.1: Functional Interfaces

6.1.3 Event-based approach

Real-life events can occur in any order, at any time, in any context. Moreover, devices are more and more mobile and can dynamically join or quit the system. In order to address this dynamicity, we made use of event-based mechanisms. Message-Oriented Middleware (MOM) offers a simple, and efficient means of communications, using the publish/subscribe principle. Event consumers subscribe to a topic they are interested in, and event producers just have to publish on the right topic. Thus, producers do not care about the presence of consumers.
To be able to use this approach, physical devices have been considered from two perspectives. *Sensors* sense real life and human actions. Their role is to feed the system with events coming from real life. They are producers of events. Consuming these events, *Actuators* act on real life, using real-life equipment. They carry out orders such as switching on the lights, or opening or closing the shutters. Components are not limited to a unique role, and can both consume and produce events.

Actuators propose two main methods. *getAvailableActions()* returns a list of actions that can be carried out on the device. If a light can answer [*on,off*], a shutter could answer [*up,down,stop*]. For each action, actuators wait for messages on a specific topic. For a sensor to ask for an action to be realized, it must know the corresponding topic. *getTopicFor(String action)* aims at providing the topic and the parameters that can be accepted for a given action in form of a *Message*.

Sensors maintain a list of messages for each event they sense. An On/Off switch maintains two lists of messages: one for each action. The messages stored also embed the topic on which they have to be published, for the action to be carried out. When an action is sensed, each message stored for this action is sent on its topic.

6.1.4 Example

For instance, figure 6.2 shows the configuration phase for the connection of a switch and a light. The Configurator retrieves the message to be sent to switch on the light.

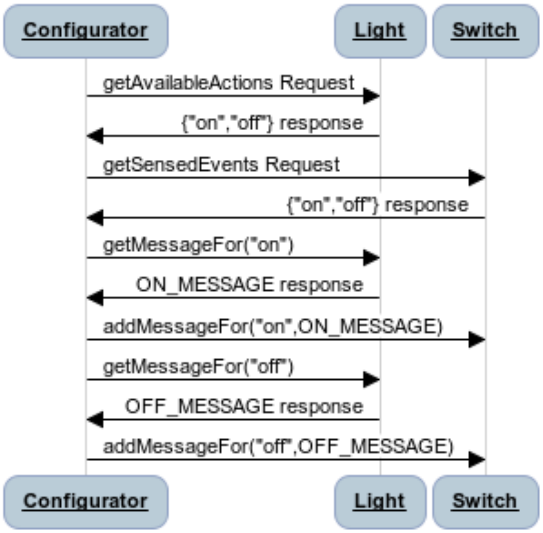


Figure 6.2: Configuration Phase

This message is added to the list of the "on" sensed value of the switch. Later, when the "on" button is pressed, all messages stocked in the list are sent. When an actuator recognizes its "on action message", it forwards the order to the real light.

This mechanism allowed us to eliminate asynchronous aspects. It also allowed any sensor to control any actuator, since they do not have to know each other to be able to work together. The action carried out and the value sensed do not have to necessarily be the same. Thanks to the mechanism of messages, it is possible to send the "on" message to the light when "down" is sensed by a shutter command.

Let us consider a home with a switch to trigger a departure scenario, operable on the KNX network. This scenario switches off all lights and closes shutters. In the considered example, there are only two shutters and one light. The first shutter was motorized when the house was built and works on a KNX network. The second shutter was added afterwards and as the owners did not want to make holes in the walls, they chose a shutter engine communicating with the command by radio frequencies. Lastly, lights are controlled by Legrand equipment. All these elements are visible on the left of figure 6.3.
The interoperation of all these elements is described in the case of a departure scenario activation. Numbers on the figure present the sequence of actions.
1- An inhabitant presses the button. This action is sensed, and generates a message on the KNX network.
2- This message is read by the driver and translated into a message for the EnTiMid system.
3- The driver then selects the virtual representation of the device responsible for the message and activates the sending of stored messages.

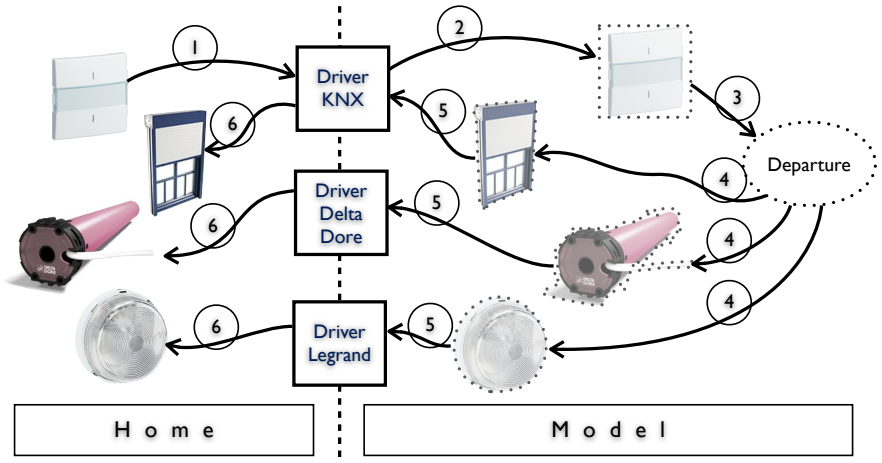


Figure 6.3: Example of Interoperability

4- Its activation causes all connected elements to be activated in parallel.
5- On receiving the message, each model representative of a real product asks its driver to send an order to the real product.
6- The driver executes the query and sends the order.
In this example, various devices with various actions are connected together. A switch that senses a *departure* is connected to two *down* actions on two different components and one *off* port. Interactions between components are possible thanks to the exchange of messages.

6.1.5 Threat to validity

Interoperability was tested and validated with a restricted set of devices. Since the main goal of this thesis is not about making any device interoperable, the study was conducted with a set of representative technologies mixing different communications media, different appliances and multiple manufacturers, in order to prove the feasibility.

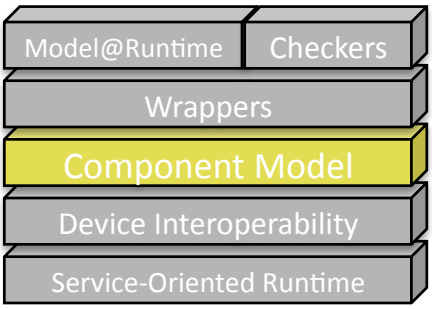
6.1.6 Summary

The *Device Interoperability* layer enables products from any manufacturer to work with any other product, in any imaginable way, by just implementing a driver. The use of this method is however limited by the non-availability of action lists at design time. Each device provides information about the actions it supports, but a method has to be called. Since method calls cannot be done at design time, the only way to obtain available actions is to go and seek them in the implementation code.
The sequence diagram, presented in figure 6.2, describes a part of the work that a piece of program has to execute to set up the application's behavior. The sequence cannot be implemented definitively, since the application may have to be adapted and to evolve

while running. The configuration has to be expressed another way, to be modifiable, and to ease the reading and understanding of the behavior.

Both of these issues require a tool. It has to be able to provide information about the devices at design time, and it has to support and help device assembly design. This tool, a new component model, is made available by the *Component Model* layer.

6.2 Component Model



On top of the *Device Interoperability* layer, a tool is required to make devices' abilities and their interactions explicit. This tool has to take into account the sporadic apparition of devices. Component models have been identified as good candidates to take on this role. They are very good representatives of real life devices, since they can use or provide services. Their interfaces(as lists of actions used or provided) are made explicit. Their life cycle is very helpful to catch the dynamicity of the devices' presence. Finally, the concept of components in software engineering is very close to electronic components. Device manufacturers and software developers have here a common discussion base, a common language.
As presented in the state of the art, component models are often too strict and prevent components from connecting with non-identical interfaces. This restriction could compromise the interoperability gained by the *Device Interoperability* level, whereas this layer just aims to simplify the configuration and management.

This section is organized as follows: section 6.2.1 emphazes the relation between the proposed component model and electronic components. This relation is illustrated in section 6.2.3. The mechanisms responsible for the synchronization of model and code are presented in section 6.2.4. Lastly, section 6.2.6 describes how the *Device Interoperability* integrates with this layer.

6.2.1 Making software components closer to electronic components

When talking about components, electronic ones are probably the first kind of component to come to mind for a lot of people. An electronic component, as shown in the bottom left part of figure 6.4, is a black box surrounded by pins. The shape of the pins is standard and allows components to be connected to any board. Neither the pins nor the board have the ability to refuse the connection of two components. This absence of constraints allows electronic components to be used in a large variety of contexts. They can be connected to a multitude of other components to create appliances. This is the perfect description of the behavior required for a software component.
Nevertheless, software components' ports are generally specialized by a programming interface (API). Thus, unlike electronic components' pins, their shapes are not standard. The goal of this specialization is to ensure the alignment of services. However, this is a too strong limitation in our context.

In electronics, components admit only three kinds of ports (pins).
Input Ports collect all necessary information from the outside, for the component to do its job. At the same time, they can trigger the execution of an associated task.

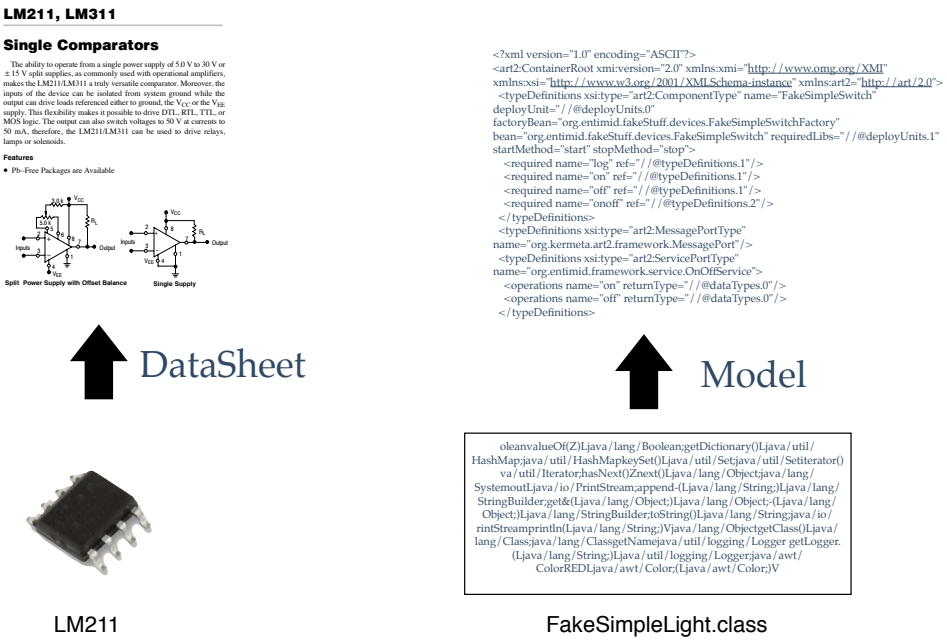


Figure 6.4: Electronic Parallel: Datasheets

Typical examples are the A and B input values of a comparator.
Output Ports release the data resulting in the execution of a task. The C result value of a comparator and the tick of a timer, are two illustrations of this kind of port.
Parameter Ports are used to set specific values for an instance. They specialize the behavior of the instance for a specific context. An example could be the clock port of a microchip, which can be set to several frequencies according to the application it is involved in.

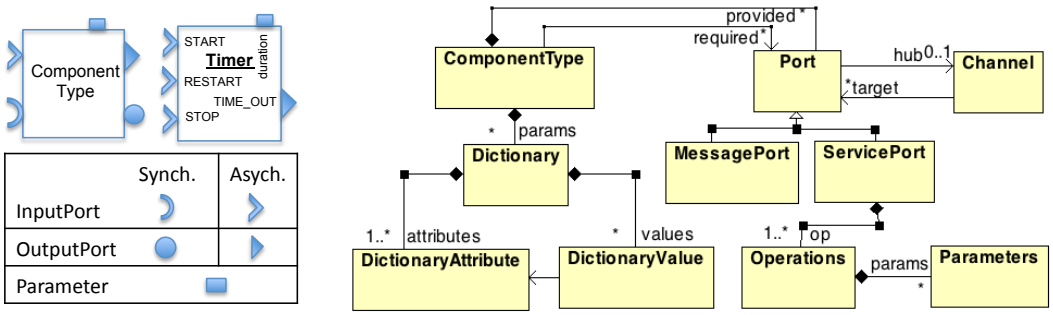
The component model created was strongly inspired by electronic components. Indeed, *InputPorts* and *OutputPorts* have been implemented as presented in figure 6.5a. They have been divided into synchronous and asynchronous kinds, to handle both object-based method call handling, and message-based communications between components.

6.2.2 Meta-Model description

This section details the elements of the component model, presented in 6.5b.

ComponentType

The *ComponentType* meta-class carries the component description specification. Ports of the component are describes in collections; provided for input ports, required



(a) Component Type (b) Meta-Model Excerpt

Figure 6.5: Extraction of a part of the component model architecture

for output ports. The component type also contains a dictionary which declares parameters, and their default values, that can be specialized for each component instance.

Ports

To keep close to existing component models and promote compatibility, the component model makes use of classical terms in its implementation. As a consequence, *InputPorts* are implemented as *provided* ports and *OutputPorts* as *required* ports, as shown in figure 6.5b. For their part *ParameterPorts* have been implemented as <key,value> dictionaries.

ParameterPorts This kind of port is used to specialize component behavior. For example, the Timer component uses a delay parameter that represents the amount of time to be spent before the time-out occurs. A component can have multiple parameters. They are uniquely named in the component's scope, and can be optional or mandatory. All parameters a component admits are listed in a dictionary at model level. At runtime, each parameter port is instantiated as a setter method, which only admits a dictionary as parameter. Indeed, each parameter port has its own setter method. Both keys and value types are Strings. This ensures the transmission of parameters in a unified way. Each component is responsible for the conversion from the String to the real type of its parameters.
To keep the link with electronic components, the method is the pin, and the dictionary describes the shape of the signal to be sent (voltage, intensity, shape of the signal).

InputPorts A component can provide several facilities to other components. This is illustrated by the Timer on figure 6.5a which offers start, stop, and restart actions. In classical software component models, the timer component would have provided a single synchronous port with the three start, stop, and restart methods. These methods

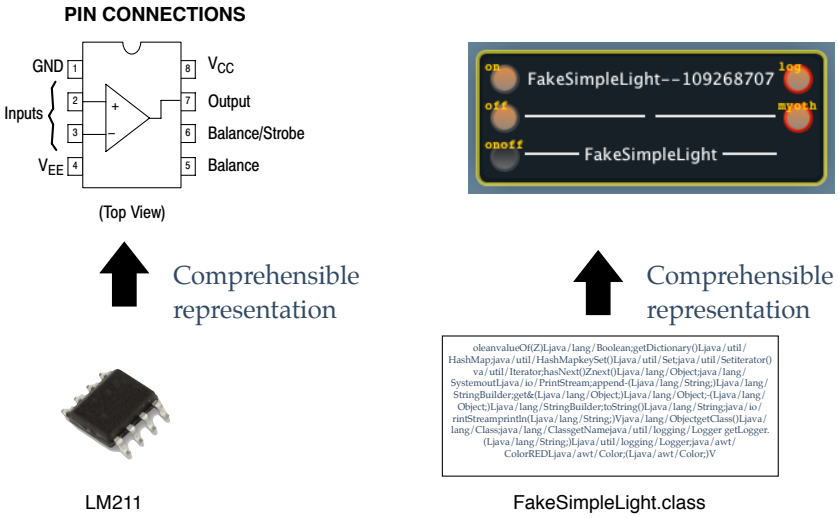


Figure 6.6: Electronic Parallel: Components

would have been defined in the StartStopRestart API.
Synchronous ports (also called ServicePorts in the model) act the usual way: they are typed by an API, and are based on method calls. Thus, our component model is able to support the common software-component behavior. However, this is not the way of designing components that we encourage. Indeed, the API is typed by the programming language type system, and this typing may prevent components from being connected because of a mismatch. We want to elevate the typing from the language to the model, and resolve the typing at a higher-level of abstraction.
Asynchronous ports, handled as MessagePorts in the component model (fig 6.5b), are much more interesting for the promotion of component connectivity. Each method/action a component offers is accessible through a dedicated port. Each port is uniquely named in the scope of the component. In the same way, electronic components have one pin for each action and actions are triggered when the value changes from 0 to 1 for instance, on the corresponding pin.
To mimic this behavior, all asynchronous *InputPorts* are implemented as a *Command* design pattern. They have a unique method *public void process(Dictionary<String, String>)*. The uniqueness and standardization of the method are mandatory to ensure the connectivity.
Just as an electronic component, actions in our component model can have parameters. Coded in the shape of the input signal passed through an input pin in electronics, our InputPorts admit a dictionary of <key, value> parameters. Like ParameterPorts, this dictionary only allows pairs of Strings. These values are specific to each execution and may change form one call to another. In figure 6.5, START, STOP and RESTART are all InputPorts. The parameters used on the start or restart activation are transferred through the TIME_OUT OutputPort to the connected component.

OutputPorts The main role of an OutputPort is to forward or release information. For instance, in figure 6.5a when the timer delay is over, the TIME_OUT port is activated and thus the connected InputPort (if any) also is. In case the activated InputPort is synchronous, the result of its activation is returned by the called method. This is a blocking behavior, and may not be adapted to events coming from real life. If the activated port is a MessagePort, the result of its execution (if any) is given though a dedicated OutputPort.

All this results in a parallel between electronic and software components as shown by figure 6.6. Input(provided) ports are displayed on the left side, and Output (required) ports are on the right.

Channel

A Channel connects two or several ports. Channels can be of different types, and are instantiated just as components are. Channels are providing communications to components. These communications can be realized with several protocols, politics and media. According to the situation, one can make use of a channel that sends messages in sequence, or of one sending in parallel. Channels can create the link using TCP or UDP sockets, RS232 serial connection, or a REST request. Channels are handling the communication semantic and method between components.
Being defined outside the component, developers can make no assumption about the media or protocol that will be used, or the kind of component on the other side. This enforces the development of well defined standalone components and promote their reuse.

Service Ports

Service ports come with the description of their operations (name, returned value) and a description of the operations' parameters (names and types). These information are used by channels, modeling tools and runtime platforms. Channels can play a role of mediation between not exactly aligned services descriptions. Modeling tools use these description to perform checks on component assemblies and authorize au not components' connections. Finally, runtime platforms can check the alignment, and may refuse any connection that violates a pre-defined connection rule.
These checks are detailed in section 6.3.1.

6.2.3 Concrete example

This example shows how a real product is implemented in this component model. The RMG4S, on the left side of figure 6.7, is a KNX product by Theben [1]. This product can control up to four 230V lights or sockets. Its virtual representative has 8 message input ports, two (i.e.: on, off) for each controllable element. When a physical event changes the state of an output of the product (somebody switches on the light using

1. http://www.theben.de/en

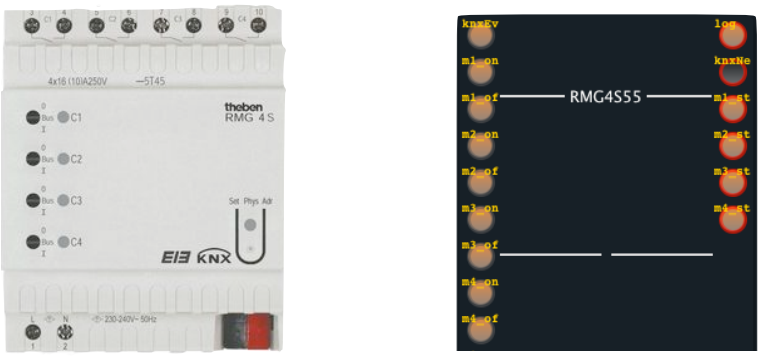


Figure 6.7: Example Model

the dedicated switch), the state is propagated to any connected device, through the corresponding port of the component.
In addition, a *KnxEnv* input port allows the driver to circulate real-life events. On the other hand, an output port *KnxNetwork* is used to send events from the model element to the real product through the driver. The last output port is a logging port.
Thus, to switch on the light that is physically connected to the first module of this product, one just has to activate the *m1_ on* port. The component then asks its driver to send a message to the real device to make it power up its first module.
On the other hand, when the state of a module is physically changed, a message is sent from the driver to the component. The component then activates the *m1_ state* (for instance), to inform any connected component about the change.
If the application proposes a graphical user interface, the *on* (resp. *off*) port of the component is activated when the user presses the graphical button. On activation, the driver sends the order to the real product, which reacts and sends information about its state change. The driver catches the information, and sends it to the graphical interface for update, through the dedicated output port of the component.

The component model makes our software equivalent to electronic components DataSheets. Assembly constraints, mandatory parameters on ports, component behavior, and many other pieces of information on components can be expressed in the model. This abstract description of the component has no effect on the runtime implementation (just as DataSheets have no effect on black-box components by the way).

6.2.4 Implementation and Model Relationship

The development of a component (i.e.: a virtual representative of a physical device) can be achieved in two ways. According to its preferences, the developer can make the model of what he wants, and ask for code generation. This approach is called *Model First*. The model can also be extracted directly from annotations decorating the implementation code made by the developer. This is called *Code First*. The code first

Listing 6.1: Java class POJO annotation

```
@Provides({
    @ProvidedPort(name = "start", type = PortType.MESSAGE),
    @ProvidedPort(name = "stop", type = PortType.MESSAGE),
    @ProvidedPort(name = "restart", type = PortType.MESSAGE) })
@Requires({
    @RequiredPort(name = "timeOut", type = PortType.MESSAGE),
    @RequiredPort(name = "logger", type = PortType.MESSAGE) })
@DictionaryType({
    @DictionaryAttribute(name = "time", default="3000") })
@Library(name="EnTiMid - Framework")
@ComponentType
public class Timer extends AbstractComponent {

  private TimerThread timer;
  private long time = 3000;  // default value

  public Timer() {}
  public Timer(final long time) { this();  this.time = time; }

  public long getTimeOut() { return this.time; }
  public void setTimeOut(long value) {
    if (value > 0) { this.time = value; }
  }

  @Port(name = "stop")
  public void stopTimer(Message m) {
    if (timer != null) { timer.reset(); }
    getPortByName("logger", MessagePort.class).process("Timer("+time+")::STOP");
  }

  @Ports({ @Port(name = "restart"), @Port(name = "start") })
  public void restartTimer(Message m) {
    if (timer != null) { timer.reset(); }
    timer = new TimerThread();
    timer.start();
    getPortByName("logger", MessagePort.class)
      .process("Timer("+time+")::STARTED");
  }

  @Start
  public void start() {
    time = Integer.valueOf(getDictionary().getValue("time")).intValue();
    getPortByName("logger", MessagePort.class).process("Start Timer");
  }

  @Stop
  public void stop() {
    getPortByName("logger", MessagePort.class).process("Stop Timer");
  }

  @Update
  public void kevUpdate() {
    time = Integer.valueOf(getDictionary().getValue("time")).intValue();
    getPortByName("logger", MessagePort.class).process("Updating Timer");
  }
```

approach differs from a reverse engineering approach, in the sense that the model is extracted from annotations in the code, and not from the implementation code itself. These two approaches are not exclusive. The model of a component can evolve, therefore impacting its implementation. Respectively, if a change is made to the code, it has to be reproduced at the modeling level. In other words, the consistency between implementation and model has to be guaranteed.

To illustrate the description, listing 6.1 shows the complete implementation class of a *Timer* component. This listing is organized as follows: on the first lines are annotations on the class that describe the component shape. Just after, the class definition comes with private attributes, object builders, then getters and setters. Next, methods render the services offered by the component. Life-cycle management methods are at the end of the class.

Component shape

The first annotations on the class inform about the *Input*, *Output* and *Parameter* ports. As explained in section 6.2.1, *InputPorts* are implemented as *provided ports*. Common actions that can be carried out on a timer (start, stop, restart) are listed under these terms.
This *Timer* implementation offers two outputs, visible as *Required Ports*. A *log* port, which sends information about the internal behavior of the component, and a *time_ out* port activated when the countdown ends.
The Timer admits a parameter. This parameter sets the delay between the start and the activation of the time_out port. This parameter appears in a dictionary.
The *@Library* indicates that the component is part of the virtual library of components called "EnTiMid - Framework". In edition tools, all components of the same library are presented under the same package of components. A library of components can be defined using several deployment units.
The last annotation tags the class as a component type implementation. This annotation is mandatory for the compilation tools to consider the class as a component type.

Port mappings

Once described, input ports have to be linked to the method implementing the action. In the example, one may remark that a port can be bound to at most one method, but a method can be reached from several ports (1 method can be mapped on n ports). Indeed, the behavior of a *start* and a *restart* of a timer are implemented the same way. Classical solutions could have been to remove one action or to copy-paste the method. From a user perspective (sect. 5.1.3), a Timer should be able to be started and restarted which implies not to removing the port. Thanks to this multiple mapping, the user will be satisfied with no redundancy of code.
Moreover, the transfer of the annotation from one method to another, changes the method called when a port is activated. This change is completely transparent for assemblies already using the component, since the annotation is not modified. This is a great ability that enables changes in the implementation and method names, with no

change in the component interface.
Finally, a component can offer the same service through both *Service* and *Message* port. Using the same mechanism, the same method can be called in both cases.

Life cycle
Start and *Stop* life cycle methods are mandatory. They are called when an instance is started (resp. stopped). Stateless components may just ignore these methods, but stateful ones may use these to persist their state.
The *update* method is used to inform a component that one of its parameters has changed.

Code first
Meta-information, concerning the component model, is introduced in the code using annotations. This method for including meta-information in the code has already been used in tools such as Fraclet[RM09]. A developer familiar with the annotation set, or in charge of the migration of existing components, may directly define the model in the code.
As in a classical development process, the new implementation code has to be compiled to incorporate the changes in binaries. Our component model takes advantage of the compilation phase to extract the model from the annotations. A visitor goes all over compiled classes and selects the *@ComponentType* decorated classes. Then sub-visitors navigate into the code to create the model.
At the end of the compilation process, the newly-computed model is added into the compilation result. *i.e.* the model is included as an XML file into the *.jar* that results from the compilation. Model consistency with the latest code version is guaranteed this way.

Model First
Writing component type code, plus the annotations, may become a complicated task. A non-familiarized person may experience difficulties in placing all the annotations required to describe its component. The model-first approach aims at providing tools to graphically (or textually) describe the component first, and then ask for the implementation to be generated. This method is made available by the use of tools such as graphical DSL, textual DSL or generic meta-modeling languages such as Kermeta [MFJ05]. Models bring a more intuitive approach for the description of a component.
Once the developer is done with its component's model, the generation tool is activated. If the implementation class of the component does not already exist, a new file is created. This file contains the skeleton of the component implementation. The code generation reaches its limits when the body of methods has to be created. The behavior of methods is the only part to be completed by hand by the developer. Otherwise, the class is already decorated with all annotations, and ports are mapped by default on generated methods.
When an implementation class already exists, the generation process is slightly more complex. In fact, a temporary model is extracted from the existing code and an Ab-

stract Syntax Tree (AST) of the code is created. The AST describes the code using a tree structure containing objects. Each object stands for a method, an argument, an attribute, etc. A comparison is made between the model created by the user, and the model extracted from the existing code. Each difference is analyzed, and modifications are made on the AST. The final code is generated from the modified AST.
The model-first approach helps somewhat in linking the model and the code. In any case, the resulting code still needs a developer to complete the new methods created, to remove dead code and to optimize the mappings.

Thanks to these mechanisms, models of components are available while the system is not running. Their conformance with the actual implementation is guaranteed by construction. This model abstraction makes it possible to create and exploit tools from MDE.

6.2.5 Implementation independence

As previously presented, the model of a component is independent from the actual implementation. Listing 6.2 illustrates this independence. Indeed, the FakeSimpleLight component declares a *onoff* provided port that renders a OnOffService. The OnOffService class is a Java interface containing two methods *on* and *off*, a kind of contract the component ensures this port is compliant with. However, the implementation class of the component does not actually implements the interface.

Listing 6.2: Implementation independence

```
@Provides({
    @ProvidedPort(name = "on", type = PortType.MESSAGE),
    @ProvidedPort(name = "off", type = PortType.MESSAGE),
    @ProvidedPort(name = "onoff", className = OnOffService.class)
})
@ComponentType
public class FakeSimpleLight extends AbstractFakeStuffComponent {
 @Ports({
  @Port(name = "on", method = "process"),
  @Port(name = "onoff", method = "on")})
 public void lightOn(Object o) {
  [...]
 }

 @Ports({
  @Port(name = "off", method = "process"),
  @Port(name = "onoff", method = "off")})
 public void lightOff(Object o) {
  [...]
 }
 [...]
}
```

Each method of the OnOffService is bind on a method of the component's implemen-

tation by mean of a *Port* annotation. Since the component does not implement the OnOffService interface, the Java compiler can not check that all methods are present. This check is thus performed while parsing the annotations for the extraction of the model, before the actual compilation.

6.2.6 Link with the interoperability layer

The actual implementation of the component model is slightly different from the view developers can have of components. The connection between two components(channel) is graphically a line linking two ports (see the top of figure 6.8). The line represent a channel, dependent of the channel type selected in the library at design time. Since ports can be synchronous or asynchronous, the runtime cannot handle port connections in the same way. The activation of an output port implies different behaviors according to its type. A message output port will send messages on topics to activate the linked input ports, but a service port has to start a method call.

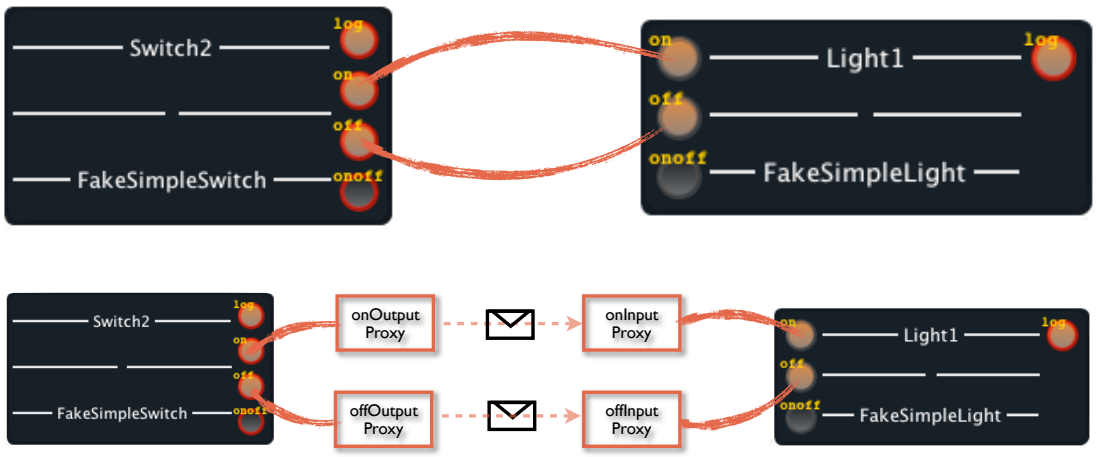


Figure 6.8: Link between the interoperability layer and component connections

This complexity is hidden from developers in both the code and the model view of the components, by the use of proxies behind the concept of channels. At runtime, a proxy is generated for each port connected to another component's port, as illustrated in the bottom of figure 6.8.
If the port is a message port, the proxies use messages and topics to communicate with each other. When an output port is activated, its proxy generates and sends a message on a pre-defined topic. On the other hand, a proxy listens to this topic only, and activates the input port on which it is connected when a message arrives. Activations are carried out with a *Command* design pattern, from the output port to activate the proxy, and from the proxy to the input port. The mechanism is thus transparent from the developer's viewpoint: an input port must provide a command pattern, an output port activates a command pattern.

The mechanism of proxies has also been implemented to handle the method calls of service ports. Links between components' ports are thus handled in a uniform way. The introduction of proxies makes it possible to use other means of communication (in the case of distribution issues for instance), and enables some adaptation mechanisms.

6.2.7 Main advantage of this component model

The main advantage of this component model is the location of the type checking. The typing of components was completely relaxed in the implementation to eliminate interoperability problems due to the implementation language type system. The typing was moved to the model level, where checks and changes in rules are made much simpler. This contribution paves the way for a pluggable type system [PACJ+08, Bra04].

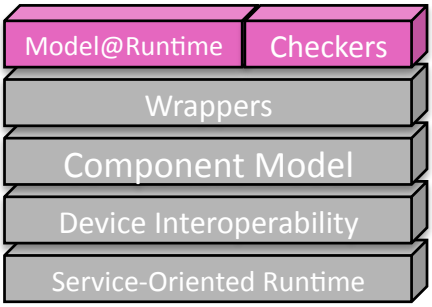
6.2.8 Summary

This new component model answers the need for a tool to make the abilities of devices and their interactions explicit at design runtime. Annotations in the code is a convenient way to integrate the component model into the implementation code, and ensure the synchronization between the model and the implementation. The component model also eases the reading and understanding of an application, since all links between components are made explicit.

The component model imposes that the configuration is completely defined, but it is not responsible for its deployment. A gap from the component assembly to the sequence of commands to set up the application at runtime still has to be filled.
Since the component model has been made flexible to allow all possible connections, any connection is possible, but some may not be desirable. Just as in electronics, assemblies are constrained by components' specificities. If electronic boards allow all possible connections, components have constraints to be respected in order to assert their behavior. Assemblies have to be verified and simulated, to prevent any undesirable interactions.

To eliminate these two issues, the Model@Runtime approach and model checkers have been used in EnTiMid. Model checkers enable verifications of component assemblies at several steps of the development, while the model@runtime takes responsibility for bridging runtime elements and the component model. These two elements of the proposal make the *Model@Runtime and Reasoning Engine* layer.

6.3 Model@Runtime and Reasoning Engine



The *Component Model* layer provides a level of abstraction from the implementation specificities. It offers a unified model view of components and their constraints and enables the creation of management tools. Reasoning engines, checkers, and models@runtime abilities can be used to ease the creation of component-based applications.
The first requirement targets the validation of assemblies, prior to their real deployment. This checking step is described in section 6.3.1. Section 6.3.2 presents an overview of the use made of Models@Runtime techniques in this approach.

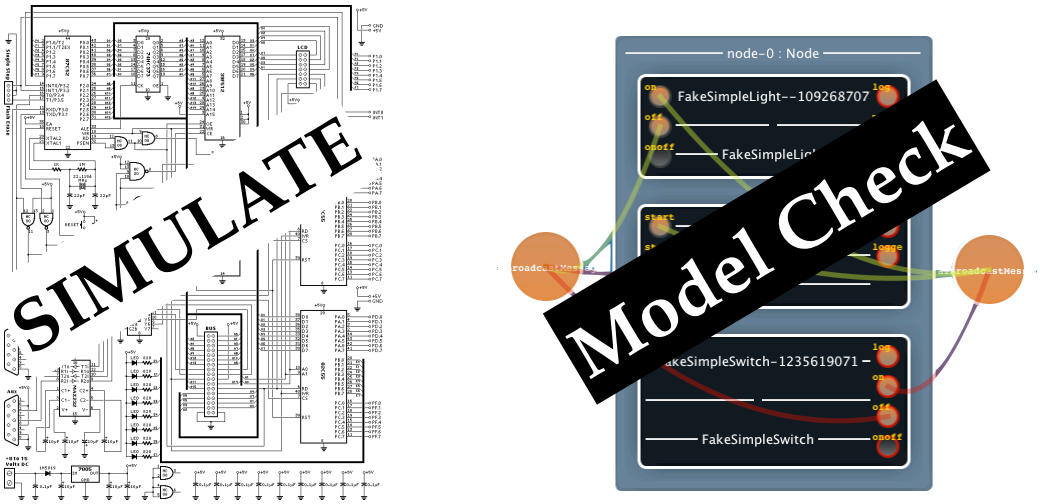


Figure 6.9: Electronic Parallel: Simulation

6.3.1 Check to validate

In electronics, the components' assembly has to be approved. Its conformance in relation to components and applications-specific constraints has to be guaranteed. This validation prevents assemblies from having any computable damage. This conformance check is often carried out by simulations, based on components' specifications described in their documentation. Figure 6.9 shows once again the parallel between the electronic approach and ours, where electronic simulations are replaced by model checking in our context.

In a classical software engineering process, conformance checking is done at 1) design time by the developer, 2) compilation time automatically, and 3) by running tests on the application built. The modeling approach offers a way to perform more precise checks, targeting more specific concerns, at several moments between the design and deployment phases.

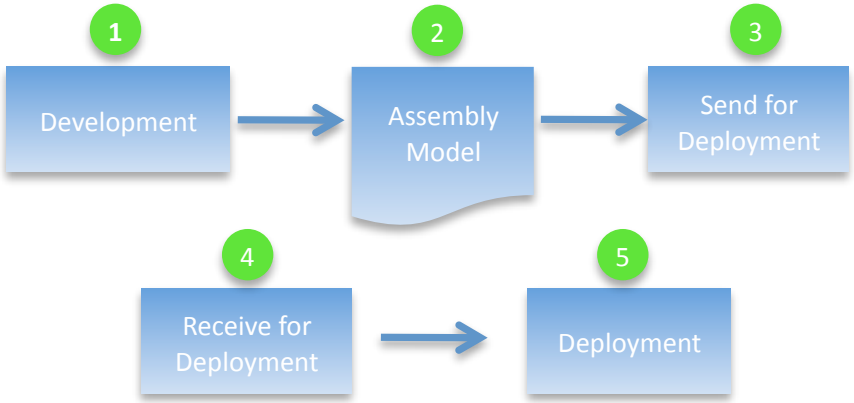


Figure 6.10: Checkpoint positions in the assembly deployment chain

Checking methods presented here have been written in Kermeta [MFJ05]. Kermeta is a modeling and aspect oriented programming language. Its underlying metamodel conforms to the EMOF standard. It is designed to write programs which are also models, to write transformations of models (programs that transform a model into another), to write constraints on these models, and to execute them. Once written, there rules have been compiled and integrated in the model editor and in the runtime platform.
Figure 6.10 displays the different moments where checks can be performed and illustrates what can be checked at each moment. These steps are detailed in the following paragraphs.

1. **Developer's actions**
   The assembly tool can monitor developers' actions. During the design phase, checkers can verify that only authorized operations are executed by the developer. When an inappropriate action is carried out, the triggering of warnings and errors can improve the development process. Thanks to this information, developers can immediately correct their code and learn from their mistakes. The earlier errors are detected and corrections made, the more the impact on the global solution is reduced. Also, developers' profiles could be created and associated with different checking policies according to the developers' expertise. This provides a fine-grained checking process for the design of applications.
   Listing 6.3 illustrates this by checking that for each channel, all connected ports are of type Service, or Message, but not both.

2. **Assembly constraints**
   The structure of an assembly can be constrained by rules, due to runtime constraints, or due to the framework used. These rules are neither specific to the developer, nor to the targeted business. A general policy could impose assemblies to be composed with at least two communication components. This constraint

Listing 6.3: Example checking developpers' actions

```
operation checkBindingsHomogeneity(model:ContainerRoot)
 : Sequence<CheckerViolation> is do
 var violations : Sequence<CheckerViolation>
 model.hubs.each { channel |
  var bindingsOnChannel : Sequence<MBinding>
  bindingsOnChannel := model.mBindings.select{mb |
   mb.hub.equals(channel)
  }
  var synchBindings :  Sequence<MBinding>
  var asynchBindings : Sequence<MBinding>
  bindingsOnChannel.each { binding |
   if binding.port.portTypeRef.isKindOf(ServicePortType) then
     synchBindings.add(binding)
   else
     asynchBindings.add(binding)
   end
  }
  if (not synchBindings.isEmpty) and (not asynchBindings.isEmpty) then
    var violation : CheckerViolation init CheckerViolation.new
    violation.message :=
    "Ports of both Service and Message kinds are connected to the same channel:"
    + channel.name
    violations.add(violation)
   end
 }

 result := violations
end
```

aims at keeping continuity in the communication service in case of failure. Valuable for all applications created by the company, this rule is shared by all software development projects.

The method presented in Listing 6.4 checks that for each component instance, all mandatory ports are connected to a channel.

Listing 6.4: Example checking assembly constraints

```
operation checkMandatoryConnections(model: ContainerRoot, node : ContainerNode)
 : Sequence<CheckerViolation> is do
 var violations : Sequence<CheckerViolation>
 node.components.each { component |
  component.required.each { port |
   if( not port.portTypeRef.optional) and port.isBind then
     var concreteViolation : CheckerViolation init CheckerViolation.new
      concreteViolation.message :=
      "Required port (" + port.eContainer.asType(ComponentInstance).name
      + "." + port.portTypeRef.name
      + ") is not bind"
      concreteViolation.addTargetObject(port)
      violations.add(concreteViolation)
   end
  }
 }
 result := violations
end
```

3. **Business Rules**
   An application created to control a plane has different constraints compared to a watering management system. Each application domain can require that special rules are considered. This checkpoint is placed just before the model deployment. The validation of conformance at this moment avoids the sending of corrupted models to the runtime.
   The rule presented in Listing 6.5 verifies that for each channel, all connected ports have the same name.

Listing 6.5: Example checking identical port names

```
operation checkPortsEquality(model:ContainerRoot)
 : Sequence<CheckerViolation> is do
 var violations : Sequence<CheckerViolation>
 model.hubs.each { channel |
  var bindingsOnChannel : Sequence<MBinding>
  bindingsOnChannel := model.mBindings.select{ mb |
   mb.hub.equals(channel)
  }
  var portName : String init ""
  bindingsOnChannel.each { binding |
   if portName.equals("") then
     portName := binding.port.portTypeRef.name
   else
    if not binding.port.portTypeRef.name.equals(portName) then
      var violation : CheckerViolation init CheckerViolation.new
      violation.message := "Connection not authorized."
   violations.add(violation)
    end
   end
  }
 }
 result := violations
end
```

4. **Platform Rules**
   A platform is a system composed of both software and hardware. The composition of the execution platform may impact the development, or deployment of a component-based application. The role of this check is to verify that all constraints inherent to the platform choice are respected. As checks are performed at the model level, they can be realized by the runtime platform itself, with no consequence on the running application. For instance, a model can be rejected if one of the components requires a serial connection, and the runtime hardware of the platform has none.
   The Listing 6.6 ensures no channel is using the SerialConnectionChannel, probably because the hardware does not have any serial port. Modeling the platform resources could enable to move these checks to before deployment and thus gain time.
5. **Check deployment commands**
   If the model of the assembly successfully passes all checkpoints, it is ready for deployment. The last step of checking consists in the first step of Models@Runtime mechanisms. The deployment of a component assembly is split into several com-

Listing 6.6: Example checking pre-deploy constraints

```
operation serialConnectionCheck(model:ContainerRoot)
 : java::lang::util::List<CheckerViolation> is do
 if model.getChannels.exists { channel |
  channel instanceOf SerialConnectionChannel
 }.isEmpty then
   var concreteViolation: CheckerViolation init CheckerViolation.new
   concreteViolation.setMessage(
    "SerialConnection not supported. (" + channel.getName + ")"
   )
   concreteViolation.setTargetObjects(channel)
   violations.add(concreteViolation)
  end
  result := violations
end
```

mands. This last verification ensures that all commands are executable before running the sequence. This phase is handled by the Models@Runtime engine. Its job consists in (1) defining the best way to go from the current system assembly to the new assembly received, and (2) supervising the migration. This is explained in section 6.3.2.

6.3.2 The Model@Runtime engine work

In his PhD Thesis [Mor10], Morin presented the concepts of a Model@Runtime engine. The Model@Runtime engine is responsible for several actions. First of all, it has to constantly maintain a model view of the running system. Secondly, when a new model is asked to be deployed, the Model@Runtime engine plans the migration (i.e.: identifies and sequences the necessary primitive commands). Lastly, it supervises the run of the migration command sequence, in order to roll back to the previous stable state in case of failure.
The next paragraphs provide an overview of the engine work, in order to ease the comprehension of solution in its entirety.

Identify and validate the changes
After validation, the first task is to identify the differences between the model representing the running system (source model) and the target model the system must switch to, as illustrated in figure 6.11. During the comparison, the next 7 types of primitive commands can be found. 1. **start** and **stop** components. 2. **add** and **remove** components. 3. **add** and **remove** bindings. 4. **update** components. The steps to go from the current configuration to the required one are specified by primitive commands that represent atomic differences between the two configuration models. The comparison system only deals with abstract commands, to allow a change of the component management policy. The real commands are instantiated (not yet executed) according to the actual policy, during the model comparison.

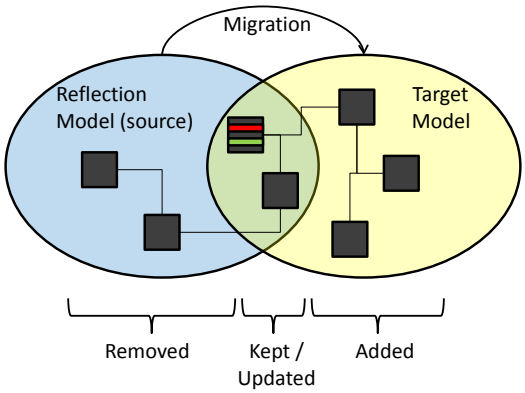


Figure 6.11: Identifying differences between the source and the target configurations.

Planning the execution sequence
These commands are stored in a collection and ordered according to a heuristic [ADN+10, BPKP10] that ensures a safe migration from the current to the target configuration. Before actually executing the commands, the list is parsed to verify that all the commands can be executed. For example, for all *AddComponent* commands, the presence of the specific component factory is checked, to ensure all components can actually be added without any problems. Doing this kind of verification for all commands ensures that the command execution will execute properly. If a command is detected as non-executable, a report clearly describes the problem, and no command at all is executed. This way, the system is always kept consistent.

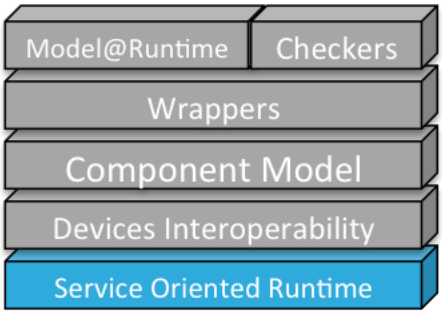
Roll-back abilities In case the migration fails, each command is decorated with a roll-back equivalent command. Thus, each command executed before the failure can be cancelled. Moreover, a second protection in place consists in keeping the old model in memory. If everything goes wrong, it is always possible to restart from scratch, and migrate back to the old model.

Specificities of components and services
Because of an adaptation, some links (bindings) between components may appear or disappear, for the system to act differently. In the case of classic components, adding or removing bindings is realized by setting or unsetting a variable. Generally, a component missing one mandatory binding is stopped, because it cannot run any longer. However, in the case of service-based systems such as EnTiMid, the component may still offer its services to third-party applications, and thus, should not always be stopped. In other words, a "light component", a virtual representation of a real light, may not be bound to any other component, but might still serve another application for the control of this light.
Other behavioral constraints can require more complex actions than just a set or an unset. For instance, if an alarm has been triggered and if the user does not process this

alarm, the system must be able to propagate the information somewhere else for the alarm to be treated. The removal of a communication link is structurally correct, but the link may take part in an operation being treated, and so, it has to be kept until the end of the action.

As previously explained, real commands for the migration are instantiated according to the current policy of the running system. Real commands can also be specialized for each runtime they have to be applied on. In our context, atomic commands have been instantiated to address a service-oriented runtime. Indeed, this runtime has offered all the facilities required by our approach. This is detailed in section 6.4.

6.4 Service-Oriented Runtime Architecture



For the proposed approach to efficiently cope with dynamic evolutions, the underlying runtime environment is required to offer dynamic abilities. As explained in [DNGM+08], the concept of service has emerged as a good candidate to cope with the dynamicity of adaptive systems. The adoption of this concept led to the development of technologies, standards, and methods to build service-based applications. Since the Service-Oriented paradigm insists on the pervasiveness of services, it imposes service-based applications to properly handle this requirement. Indeed, services can appear and disappear at any time, and applications built upon these principles have to take these constraints into account.

The OSGi Alliance [All11], a 'consortium of technology innovators', has released a set of specifications that define a service-oriented platform, and its common services. This Service-Oriented Runtime has been selected to support commands that require adding or removing component instances and types(binaries), during the execution.

Dynamicity in OSGi
The OSGi kernel is a standard container-provider to build service-oriented software systems. It implements a cooperative model where applications can dynamically discover and use services, provided by other applications running inside the same kernel. It provides a continuous computing environment. Applications can be installed, started, stopped, updated, and uninstalled, without a system restart. It offers a remote management model for applications that can operate unattended or under the control of a platform operator. Finally it embeds an extensive security model, so that applications can run in a shielded environment. According to these specifications, an application is then divided into several bundles. A bundle is a library component in OSGi terms. It packages services that are logically related. It imports and exports Java packages, and offers or requires services. Services are implementations of Java interfaces.

Modularity
Each OSGi bundle is designed to reach the highest level of independence, giving the software enough modularity to allow partial service updates, additions or removals. This programming style allows software-builders to deploy the same pieces of software for all of their clients, either professionals or private individuals and then simply adapt the services installed. Moreover, the services running on the system can be changed during execution.

Component Types, Instances and Bundles

Described in section 6.3.2, the Model@Runtime engine creates an ordered sequence of commands when it receives a new model to deploy. Each command of the list is then translated into an OSGi command.
In EnTiMid, component types are contained in OSGi bundles. These bundles are only used as deployment units and do not provide any service. They just embed components. When an **addComponent** command is parsed, the runtime checks if the component type is available in the environment. If not, the bundle containing the type is downloaded and installed. Once the component type is available, a new instance can be created.
Component instances are also mapped on bundles because of their independent life-cycle management. Indeed, **start** or **stop component** commands are directly translated to start and stop bundle OSGi commands.

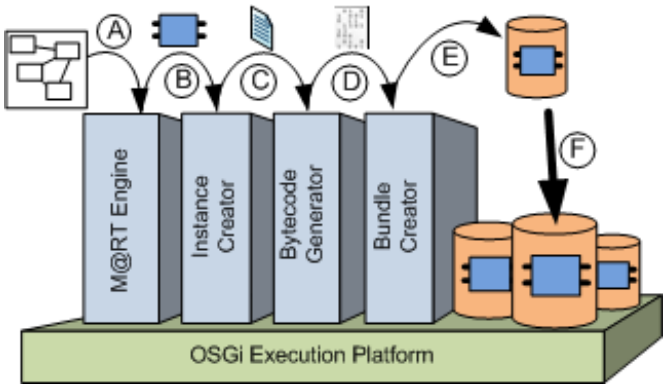


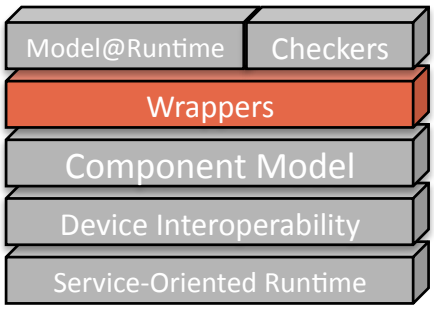
Figure 6.12: Instance creation tool chain

The creation of an instance is realized as presented on figure 6.12. From the model (step A), a new instance is queried by the Model@Runtime engine(step B) to an instance creator. This instance creator can be a Java code generator, an XML generator, or whatever. The instance creator then asks for the bytecode generator to compile the instance. This compilation can be realized with ASM [2] for Java code, handled by Spring for an XML file, etc. Step D consists in packaging the bytecode in a bundle. Step E makes it available for the runtime platform. The last step installs the instance bundle in the system.

Mapped on OSGi bundles, component instances can offer services to other components in the component model, or to other bundles on the OSGi platform. This facility makes it possible to dynamically expose instances on application-level protocols. This role is supported by the *Wrappers* layer described in section 6.5.

2. http://asm.ow2.org

6.5 Wrappers



The wrappers layer takes advantage of the component model layer, which makes it possible to dynamically and automatically wrap devices into several current and future application level protocols. For instance, UPnP, DPWS or Digital Living Network Alliance (DLNA) are such kinds of protocols. Their implementations is often too heavy to be implanted into the devices themselves. The role of this layer is thus to export all devices for free, on several protocols. Rather than offering an automatic publication mechanism to a selected protocol, the wrappers layer offers a means to publish any component to any protocol. Each component can thus be accessible using as many protocols as there are wrappers. This approach has been presented in [NDBJ08].

Component Model *versus* **Third party**

Wrappers need to obtain information about the devices present in a given deployment. This information can be retrieved in two ways.
(1) The wrapper is designed as a component. As a consequence, it can monitor the current model of the running system by asking the *Model@Runtime* layer. Any change in the model implies that the wrapper check new devices and removed ones. Another benefit of this approach is that a wrapper is considered as a classical component. The model can thus manage it as any other component.
(2) The wrapper is built as a third party application, running on the same platform. In this case, the wrapper monitors registrations of services in the OSGi context. Each time a device registers a service, this service is made accessible through the protocol handled by the wrapper. This approach has two drawbacks. Firstly, the export and use of services are not visible in the model. A device can thus be removed while in use through an application protocol. Secondly, the life cycles of the services exported on the application protocol depend on the registration and unregistration of the device's services. Since no dependency is expressed in the model, the life-cycle management of exported services has to be handled 'by hand' by the wrapper.

Reversed drivers

A wrapper is created for each application level protocol. Just as for devices' drivers, the deployment of a wrapper is required for each application protocol to be addressed. Each wrapper monitors the current application to detect addition or removal of devices. For each device, the wrapper takes on the role of a proxy and handles communications to and from the application level protocol.

6.6 Summary

The model used by the Model@Runtime engine has been augmented with the introduction of the component model described in section 6.2.
The service-oriented architecture of the runtime makes it possible to cope with evolutions and adaptations, since bundles can be installed and updated with no need to restart the system. These facilities are exploited by the Model@Runtime layer, which comes with tools and methods to address the evolutions, variability, adaptations and safety of the application. However, these properties could not have been used without the creation of a new component model inspired by electronic components. This component model improves flexibility and enables the connection of heterogeneous components, while keeping a high level of reliability thanks to checkers. The Device Interoperability and Wrappers layers provide abstractions of manufacturers' specificities and free publications on application level protocols respectively, to promote interoperability and openness.

Chapter 7

Outcomes

This chapter aims at providing a bit more details about the outcomes of this thesis in terms of implementation and tools. The first section gives quite general information about the implementation of this component model and provides some metrics. The second and last section of this chapter classifies the component model according to the classification proposed by Crnkovic in [CCSV07].

7.1 Implementation

The ideas presented in this thesis have been tested and improved in several situations and contexts thanks to an implementation.

EnTiMid has grouped together all separated layers to provide a first running implementation of the component model and tools. Because of the context, EnTiMid targets the creation and realization of software applications in the domain of Home Automation. However, ideas presented in this thesis are more general and applicable in other domains such as Machine-To-Machine or automotive industry.
The arrival of a new PhD student in the team incited to rethought and rebuild the software solution.

Kevoree has inspired and integrated ideas developed in this thesis and in EnTiMid, to create a more general purpose software development environment. This generalization enables the use of these ideas in several projects and implementation in the team. It also allows for future specializations or extensions of this work. As for EnTiMid, it has been re-factored to specialize Kevoree for a use in the domain of HomeAutomation.

EnTiMid concentrates in integrating new ideas like Models@Runtime and the flexible component model. Thus, an iterative process on top of an OSGi platform led to the creation of a new component model with its implementation. However, this implementation allowed us to change, drop, refactor our code to validate our ideas, which could have been more difficult by extending existing platforms.

If the implementation is home-made, the structure of components in our component model is close to existing models, making it possible to develop extensions to integrate the contributions of this thesis to existing models.

7.2 Impact on the development process

The use of EnTiMid impacts the development process at several levels. To describe this impact, this section details the elements manipulated and the tools available for different development phases.

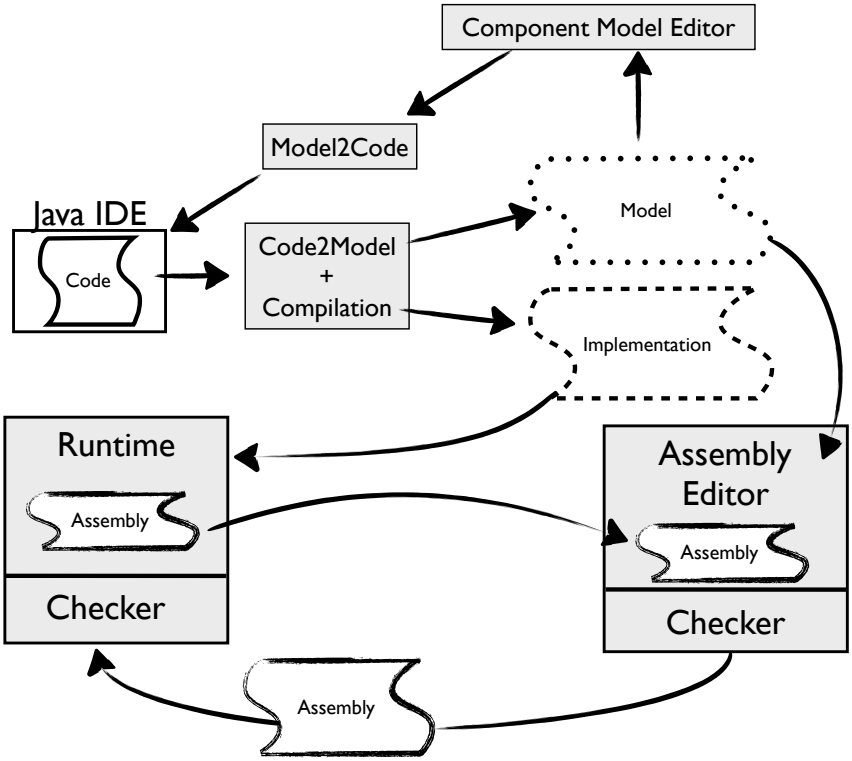


Figure 7.1: EnTiMid development chain

7.2.1 Component development

Since the component model is highly permissive, component developers cannot make any assumption on the context of use of the component. Components are thus required to be well protected against wrong usages, bad or missing parameters.
The use of annotations to describe the model and map ports and methods, makes easier the migrations of code, refactorings and evolutions of implementations. The model of a component described by the annotations can be different from its actual imple-

mentation, because several ports can be mapped on a single method. Moreover, if a component offers a service port (implementing a specific interface), the component's implementation class does not have to (but still can) implement the interface of the provided service. The only required thing is to map all methods of the service interface to operation in the code.
These mappings provide flexibility to component models and their implementations.

The compilation chain, represented on figure 7.1 by the "Code2Model" box, automatically checks for mandatory life-cycle methods and verifies that all methods of all ports are actually mapped on implementation methods. In the same time, the chain extracts the models of the components by parsing the annotations. These models are then embedded in the archive containing the sources compiled in the same compilation process. This guaranties the model-code consistency by construction.

As presented on top of figure 7.1, a component editor can be used to create or modify an existing component model. This model of component is then synchronized with the code thanks to a tool, "Model2Code engine", which generates a component's implementation skeleton or updates the annotations in an existing code.

7.2.2 Application design

In a design process, the component model of EnTiMid and the assembly tools make it possible to connect heterogeneous components, such as actual devices, functions, or services available through internet. Since the model is very flexible and permissive, designer can try and deploy any combination of components, unless checkers in the assembly tool or in the runtime reject the model. The bottom of figure 7.1 illustrates this.

According to the domain of application, checkers can be specialized to avoid assemblies identified as leading to failures. Moreover, checkers can be adapted to the current user of the application and give more authorizations to an engineer, and more restrictions to the end-user for instance.

Finally, runtime checkers have to be specialized to reject models requiring resources not available on the platform (serial port connections for instance).

In case an adaptation or an evolution is required, the model of the running system is always available and can be collected directly from the runtime. Once updated, the model of the system can be sent to the runtime for a adaptation of the actually running application.

7.3 Metrics

The table 7.1 provides some lines of code metrics of Kevoree and EnTiMid. These metrics have been measured on Kevoree 1.5.0-SNAPSHOT and EnTiMid 3.0.0-SNAPSHOT

on the 23rd November 2011. The counting was realized using Cloc [1] by considering only files contained in the *src* folder of each project. Thus generated code has not been counted. A Maven [2] plugin recursively launched the counting on sub-modules of a top-project, and aggregated the results.

Project	Scala	Java	Total	Part	Value
Kevoree-Core	8143	344	8487	11,54%	979,3
Kevoree-Tools	8483	6255	14738	22,22%	3274,65
Kevoree-Extra(ecore)	1275	126	1401	50%	700,5
EnTiMid-Core	0	445	445	85%	378,25
EnTiMid-Tools	82	315	3197	75%	2397,75
EnTiMid-Library	0	2065	2065	80%	1652
EnTiMid-Extra	0	1091	1091	85%	927,35

Table 7.1: Lines of code metrics of Kevoree and EnTiMid

Kevoree and EnTiMid have been implemented in mixed Java and Scala languages. The table provides an indication of my participation in the different projects of the implementation. The complement has been realized in collaboration with other PhD students of the team.
The core of Kevoree embeds the implementation of the meta-model, a loader, a serializer, a cloner and some utility classes grouped in a framework used for both runtime management and design of components.
Kevoree-Tools contains a graphical editor for components and assemblies, model and code synchronization tools and the definition of a scripting language to ease the management of models.
Since, Kevoree can be seen as a generalization of EnTiMid, the most important part of development in EnTiMid was paid in creating component libraries and extra APIs wrapping, specialized for home automation technologies and devices. For demonstration purposes, several additional tools had also been realized.

7.4 Classification

Crnkovic proposed in [CCSV07] a classification framework "to increase the understanding of concepts and easier differentiate component models". This section aims at classifying the component model of this thesis according to this classification. Four dimensions are considered in [CCSV07]. Each dimension is presented in a separated subsection.
The tables presented here have been extracted (for existing component models) from the Crnkovic classification, and completed to integrate EnTiMid. The component models selected for this extraction are also present in the state-of-the-art section.

1. http://cloc.sourceforge.net/
2. http://maven.apache.org/

7.4.1 Lifecycle

Component Models	Modeling	Implementation	Packaging	Deployment
EJB	N/A	Java	EJB-Jar files	At run-time
Fractal	ADL-Like Language(Fractal ADL, Fractal IDL) Annotations(Fractlet)	Java(in Julia, Aokell) C/C++(in Think) .Net lang(in FracNet)	File system based Repository	At run-time
OSGi	N/A	Java	Jar-files (bundles)	At run-time and at compilation
EnTiMid	Graphical and Textual ADL, Annotations(CDL)	Java, Scala	Jar-files(bundles) Repositories	At run-time and at compilation

Table 7.2: The Lifecycle dimension

EnTiMid proposes several tools for editing components and applications. Developments can be done in Java and/or Scala, and deployments (based on OSGi mechanisms) are realized using Jar files and repositories. This deployment can be done while the system is running.

7.4.2 Constructs

Component Models	Interface type	Provides/Require distinction	Distinctive Features	Interface Language	Interface Levels
EJB	Operation-based	No	N/A	Java Programming Language + Annotations	Syntactic
Fractal	Operation-based	Yes	Component Interface, Control Interface	IDL, Fractal ADL, Java, C, Behavioral Protocol	Syntactic, Behavior
OSGi	Operation-based	Yes	Dynamic Interfaces	Java	Syntactic
EnTiMid	Port-based	Yes	Types Specified in Interfaces	Java, Annotation, other	Syntactic, Semantic

Table 7.3: The Constructs dimension - Interface Specifications

Each port in EnTiMid can be specified independently, making interfaces types port-based. The model also differentiates between provided and required ports. The types of ports are specified in the component model and their parameters also, which is a particularity. The language to describe the interface can be Java(for services ports) or just annotations (for message ports). In case of message ports, they handle the semantic of the call; otherwise it is a syntactic interface level.

Interaction style between components depends on the channel used. The channel supports the interaction and can act as a blocking service call, as a trigger or even with some callback mechanisms. Thus communications can be synchronous or asynchronous according to the channel chosen. Bindings are made explicit by the concept of channels.

Component Models	Interaction Styles	Communication Type	Binding Type	
			Exogenous	Hierarchical
EJB	Request Response	Synchronous, Asynchronous	No	No
Fractal	Multiple Interaction Styles	Synchronous, Asynchronous	Yes	Delegation, Aggregation
OSGi	Request Response, Triggering	Synchronous	No	No
EnTiMid	Multiple Interaction Styles	Synchronous, Asynchronous	Yes	Delegation

Table 7.4: The Constructs dimension - Interaction

In case of a composition, the behavior of the binding is dependent of the implementation of the composite component. As of today, the unique composition mechanism available in EnTiMid is the inheritance, and can be considered as a delegation in this case.

7.4.3 Extra-Functional Properties

Component Models	Management of EFP	Properties Specification	Composition and Analysis support
EJB	Exogenous System wide(D)	N/A	N/A
Fractal	Exogenous per collaboration(C)	Ability to add properties (by adding "property" controllers)	N/A
OSGi	Endogenous per collaboration(A)	N/A	N/A
EnTiMid	Exogenous System wide(D)	Metrics<key,value> values updated at run-time	Handled by pluggable reasoners

Table 7.5: The Extra-Functional Properties dimension

Working with a model at run-time makes it possible to dynamically get the value of a metric on a running component. Metrics are specified on and updated by components in the model. Theses metrics are <key,value> pairs and can represent any interesting data on the component behavior. Their analysis can be performed locally or remotely by reasoner capable of making a decision from a given model extracted from the running system.

7.4.4 Domains

Component Models	General Purpose	Specialized	Generative
EJB	X		
Fractal	X		X
OSGi	X		
EnTiMid		X	X

Table 7.6: The Domains dimension

Dedicated to the domain of Home Automation and AAL, EnTiMid is clearly specialized. Kevoree, on its side, can be considered as more generic.

Part III

Validation

Winners compare their achievements with their goals, while losers compare their achievements with those of other people.
Nido Qubein

EnTiMid was tested on a realistic use-case scenario, in which all previously-listed properties were stressed. This scenario, defined in collaboration with partners of an AAL project, is presented this chapter 8.

Chapter 8

Validation in the context of an AAL project

As one of the IDA project's partners, we proposed EnTiMid as a flexible integration tool, for the different types of equipment offered by industrialists. EnTiMid has been considered and evaluated on a scenario, collaboratively defined with the partners of the project. It has been designed to underline the properties required for such a system.

The first section of this chapter introduces the project and its context and details the scenario used as a validation for EnTiMid. Sections 8.4 to 8.7 describe the evaluation of a set of properties: the environment setup, the procedures, and the results for an evaluated property. Section 8.8 lists some threats to the validity of this study. The conclusion and perspectives of these experiments is presented in section 8.9.

8.1 Context of the study: the IDA project

The first phase of the Innovation Domicile Autonomie (IDA) [1] project took place from June 2008 to June 2010, in the City of Rennes and the greater Rennes area(France). This local AAL project was funded by the Regional Council of Brittany to investigate issues resulting from the ageing of the population, and its socio-economic impact. More precisely, the IDA project involved conducting an inquiry about the use of ICT to help elderly people in their everyday life at home. To this end, the project involved:

Association for care at home ASSAD du Pays de Rennes
Industrialists Custos, Delta Dore, Urmet Captive, Spartime, i–Pocarte, Domtis, Ordimemo, Intervox Systems, Laudren, Solem
Social housing authority Archipel Habitat
Installers Marsollier Domotique, Lepage Electronique
Project ownership assistance ARELIA

1. http://www.ida-autonomie.fr/

Research institutes INRIA (National Institute of Research in Computer Science and Control), the University of Rennes 1, the University of Rennes 2, LOUSTIC (Information and Communication technology Observation Laboratory), IETR (Rennes Electronics and Telecommunications Institute), CRPCC (Psychology, Cognition and Communication Research Center)

Public administrations Rennes Metropole, Conseil General d'Ille et Vilaine, Ville de Rennes, CCI Rennes, Critt Santé, MEITO

Elderly people Anonymous individuals for tests.

The conclusions of the project have been compiled in [ASS10].

The "ASSAD du Pays de Rennes" Association, leader of this project, employs 450 persons among which, nurses, home care assistants, technicians, etc. The association helps more than 3,000 persons, in 17 towns in the south and east of Rennes.

Although the IDA was not funded by the European AAL program the assisted (elderly) person was project-centred, and the six dimensions described in section 2.1.2 appeared in the background all throughout the project. The ASSAD association took care of this point.
EnTiMid had been presented as a system for the integration of the several devices evaluated in the project. Indeed, the objective of IDA was to measure how ICT could foster the autonomy of elderly people at home and support the activity of carers. In this context, we proposed EnTiMid as an integration platform, able to deal with various devices and services, and promote the deployment of solutions adapted to each person's needs.

Interested in the proposition, collaborative work had been engaged within the IDA project. With the help of the ASSAD, Delta Dore, Custos, Arelia, and the LOUSTIC, among the most active, a scenario was defined to put EnTiMid in context.
This case study was designed to be as close as possible to real life conditions. The story was introduced by the ASSAD association. Products were proposed by Delta Dore. The evolutions were gathered from past experiences of the ASSAD, Custos and the LOUSTIC. This scenario was set up to evaluate EnTiMid on a realistic case, with the same devices as those actually deployed. It stresses several issues to measure how EnTiMid would cope with them.
The following part of this section presents the story and the evaluation context of EnTiMid. The way EnTiMid addressed the problems is described for each issue in a separate section.

8.2 Use case and issues to address

The scenario used for the evaluation of EnTiMid is presented in this section. It involves an elderly person called Mrs P., several members of her family, and different devices selected in collaboration with other partners in order for the evaluation to be

realized with real products and in the closest conditions to reality.

The scenario is presented here with names of persons and real products for the sake of clarity.

The scenario is about Mrs P. She is seventy-eight. She has two children and five grandchildren. Mrs P. begins to experience some difficulties in walking and moving. To improve her safety, her daughter proposed that she moves to an equipped flat.
Among other equipment, the flat is basically equipped with an alert system, which triggers a voice call to a care center when a button is pressed on a remote control. But Mrs P. is not comfortable with this equipment and would prefer a more generic remote control. She also would like the alert to be sent to her daughter, instead of to the care center.

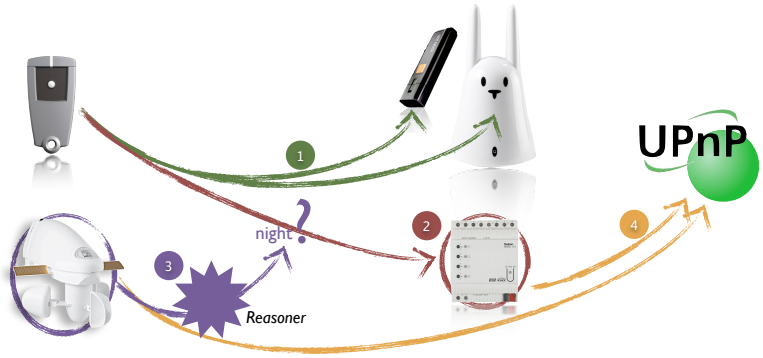


Figure 8.1: Solution elements for Mrs P.

The remote control, on the top left corner of figure 8.1, is a one-button command from Delta Dore (a French manufacturer of home automation devices). This remote control has been designed to be universal for any receiver product from the Delta Dore catalog. A 3G-communication stick (Icon 225) is used to send alerts to Mrs P.'s daughter. Then a Nabaztag rabbit helpfully provides feedback to Mrs P. when she asks for help from her daughter.
The connection of these three items raises **interoperability** problems (symbolized by the bullet number one), which are detailed and answered in section 8.4.

After some months of use, Mrs P. asks for the system to automatically switch on the lights, when an alert is sent. Luckily, the flat is already equipped with devices that enable the control of lights. The light control is made available by a RMG4S device from Theben (on the bottom right of figure 8.1, working on a KNX network. Section 8.5 presents how EnTiMid enabled this **evolution**.

Following this evolution, the behavior of the lights was sub-optimal, because the lights were switched on whatever the period of the day. To eliminate this issue, an **adaptation** mechanism(bullet 3), described in section 8.6, was deployed. The sensing

of daylight was realized by a KNX weather station outside the house. This weather station is visible on the bottom left corner of figure 8.1.

One day, her son came with a new touch screen device. This touch screen would allow Mrs P. to easily access the Internet and have video calls with her children and grandchildren. The touch screen is able to control devices over UPnP and DPWS networks. In order to allow the control of home devices with this touch screen, the solution deployed is required to be UPnP and/or DPWS compliant. This requirement stresses the need for **openness**. Section 8.7 elaborates about the mechanisms used to answer the fourth bullet of figure 8.1.

In order to evaluate the answers of EnTiMid in this scenario, and since a real deployment cannot be realized, a test environment with real devices was set up. Just before the description of the solutions offered by EnTiMid, the different elements composing this environment are presented in section 8.3.

8.3 Experimental setup

The test environment of this study was realized with equipment provided by industrial partners of the IDA project for one part, and funded by the HID platform, financed by a European Regional Developments Fund for the other part. It is mainly composed of two home mock-ups, one with Delta Dore devices exclusively, the other with KNX devices exclusively and an MSI Top touch-screen PC, as visible in figure 8.2.



Figure 8.2: Equipments available for the study

8.3.1 Delta Dore equipment

The Delta Dore mock-up, visible on the left side of picture 8.2, was set up with devices from heating, alarm, security and automatism functional domains. Here is the description of all the elements.

- A *DRIVER 210 CPL + TYDOM 520* heating controller, on the bottom left corner of the mock-up, controls two heater receivers: a *TC51089* PLC receiver, and a *RF660FP* radio receiver.
- The big black box on the top right corner of the mock-up is a *TYXAL CSX40* alarm, which collects information from several sensors. A *DOFX* smoke sensor, and two *MiniCOX* door sensors, visible on the left side of the alarm. The last sensor is a *DFX* water leak sensor (green object on the table, under the mock-up).
- One *TYXIA 442* light dimming transceiver, and a *TYXIA 411* timed power switch, both hidden behind the rabbit.
- Not visible in the picture, two remote controls. One *TYXIA 110* with a single ON/OFF button, and a *TYXIA 141* with four.

All these elements use the X2D protocol, owned by of Delta Dore, to communicate on both PLC and radio media. Since the protocol is not public, Delta Dore lent us a research and development product, able to communicate in both ways (listen and act) on the X2D network.

8.3.2 KNX equipment

The second mock-up, on the right side of the picture, is made of KNX compatible products only, mainly from the manufacturer Theben and some from Siemens.

- On the top right corner of the KNX mock-up is an outdoor weather station. This weather station gives information about the wind, the rain, the temperature, and the light value. This information can be provided whether periodically or when a value changes slightly.
- Just under the weather station, on the left, the *LUNA 113* is a light sensor for the outside. On its right, an *AMUN 7160* provides information about temperature, humidity and CO_2 rates inside the house.
- Going down the right side comes the *VARIA 826 WH KNX*, which is an ambient controller. It allows for changes in the heating regulation values, reading information from the weather station and many other appliances.
- The four switches, in the bottom right corner of the mock-up, are controlled by a *TA 4*.
- The electric panel includes three other devices. An *RMG4s* device controls the four power sockets, at the very bottom of the panel, with a On/Off behavior only. Above this device, a DMG 2 controls the dimming of the two sockets on its right.

– Hardly visible on the top of the panel, a Siemens *EIB/IP N148/21* gateway makes it possible to access the KNX network through an IP connection. Helped by the Calimero [2] framework, it enables a programmatic control on all the devices, and allows the user to listen for events on the KNX network.

8.3.3 Other equipment

The link between all of these devices is made by EnTiMid, but it still requires an environment for its execution. Several other devices are available in this experimental setup.

– An all-in-one PC *MSI Wind Top*, with a touch screen, an Intel Atom 230@1.6GHz CPU, 1Gb RAM and Ubuntu 9.10 (Linux kernel: 2.6.31-17-generic) for the operating system.
– An *Icon 225* 3G USB modem, used only for sending short text messages
– A Nabaztag:tag, the big rabbit in the picture, able to synthesize a voice from a text, and used to provide feedback to the user. A Nanoztag with a Mir:ror (little grey rabbit and round blue base) are used as an Radio Frequency IDentification (RFID) tag and an RFID reader respectively.
– An Ethernet router to connect the KNX mock-up and the touch-screen.

8.4 Interoperability issue

The first problem that the story of Mrs. P emphasises, is the connection of three heterogenous devices.

8.4.1 Test Environment

To evaluate EnTiMid on this issue, we made use of the Tyxia 110 remote control from the Delta Dore mock-up, the 3G modem to send text messages and the Nabaztag:tag rabbit to provide feedback to the end user.
Nothing in EnTiMid was already available to access these products.
The MSI Top touch-screen was used for the deployment of the test.

8.4.2 Resolution Protocol

Driver creation
Each product used in this test is from a different manufacturer. Thus, three drivers had to be created, as described in section 6.1.1 of the contribution.

The driver enabling the use of the Tyxia 110 makes use of the gateway offered by the manufacturer, making it possible to listen on the X2D network. This gateway was delivered with a Java API, which simplified the creation of the driver. Indeed, the driver just consists of the creation of a listener and of a class to handle the implementation and

2. http://calimero.sourceforge.net

the model of the remote control device. The Tyxia 110 component has a unique output port *pressed*, and can be customized to specify the parameters to be sent through the output port when the button is pressed.

The second element, the 3G modem was considered as a simple modem. The sequence of *AT* commands to be sent to the modem, to send a short text message, was collected from the modem documentation. With the help of a serial communication library in Java (RxTx), the component was implemented, and decorated with modeling annotations. The Icon 225 component representative offers a unique input port *send*. This port admits one parameter: the text of the message to send. The receiver's phone number is given as a parameter to the component.

The Nabaztag:tag rabbit is the last element in this test and is used to provide feedback to the final user. The web-service API of the Nabaztag rabbit provides a Text-To-Speech facility that can generate and return an MP3 file. Indeed, the rabbit is able to either, directly synthesize a voice from a text or generate a file containing the voice synthesis, for it to be played later by the rabbit. The component standing in for the rabbit thus proposes a *generate* input port. The action of this port is to call the text-to-speech facility with the text passed through the port as a parameter. The generated MP3 file is then returned through an output port called *generated*. A second input port, *play*, can be used to ask the rabbit to read a text or an MP3. If a text is given in parameter, the synthesis is made on the fly.



Figure 8.3: Components used in the interoperability experiment

Connection of the elements

An instance of each element of the assembly is created in the model. The Tyxia 110 is customized to send two parameters through its output port on activation: the text "Your request has been sent to your daughter." for the rabbit, and "Your mother is requesting a call from you." for the Icon 225.
This message with the two parameters is forwarded to a dispatcher, which triggers in parallel the *send* input port of the Icon 225 instance, and the *play* input port of the Nabaztag. Each component collects the parameter it is interested in and carries out the required action.

8.4.3 Results

At first sight, the connection of a generic remote control, an electronic rabbit, and a short message modem is not that obvious. Industrial partners of the project were baffled by this requirement, because it implied that they develop an ad-hoc device. This is not viable when targeting a provision of specialized solutions for each person's needs.

In this example, three drivers had to be developed to be able to get components in the model. Each driver can now be augmented to provide more products from each manufacturer. The development of drivers, once done, is never to be done again. Components can also be reused in other contexts, because of their independence and the avoidance of direct connections. For future applications, this signifies a great gain in time.

8.5 Evolution issue

The evolution in this use case is due to a change in Mrs P.'s needs. She wants the lights to be automatically switched on when she presses the remote control. This requirement is related to a feeling of safety when lights are on at night.

8.5.1 Test Environment

This evaluation makes use of the previous devices involved in the interoperability evaluation.
In addition, the RMG4s of the KNX mock-up is integrated in the application as illustrated in figure 8.4.

8.5.2 Resolution Protocol

No KNX product had been used for the moment. So, the first task was to create the driver to control KNX equipment and obtain a model representative for the RMG4s (presented in figure 6.7 a few pages back). Once the component is ready, it has to be deployed.
Thanks to the Model@Runtime layer, the technician responsible for the addition of the new functionality retrieves the current model of the running application, using a TCP/IP remote access. He then adds an instance of RMG4s and connects all *on* input ports to the dispatcher already present. In this configuration, all lights controlled by the RMG4s are lit when the Tyxia 110 button is pressed, in addition to the text being sent and the rabbit speaking.

For the last step, the technician sends back the model to the runtime of EnTiMid in the home. Once all checks are passed, the runtime downloads the newly-created component type and its driver, in order to create and connect the instance. All this

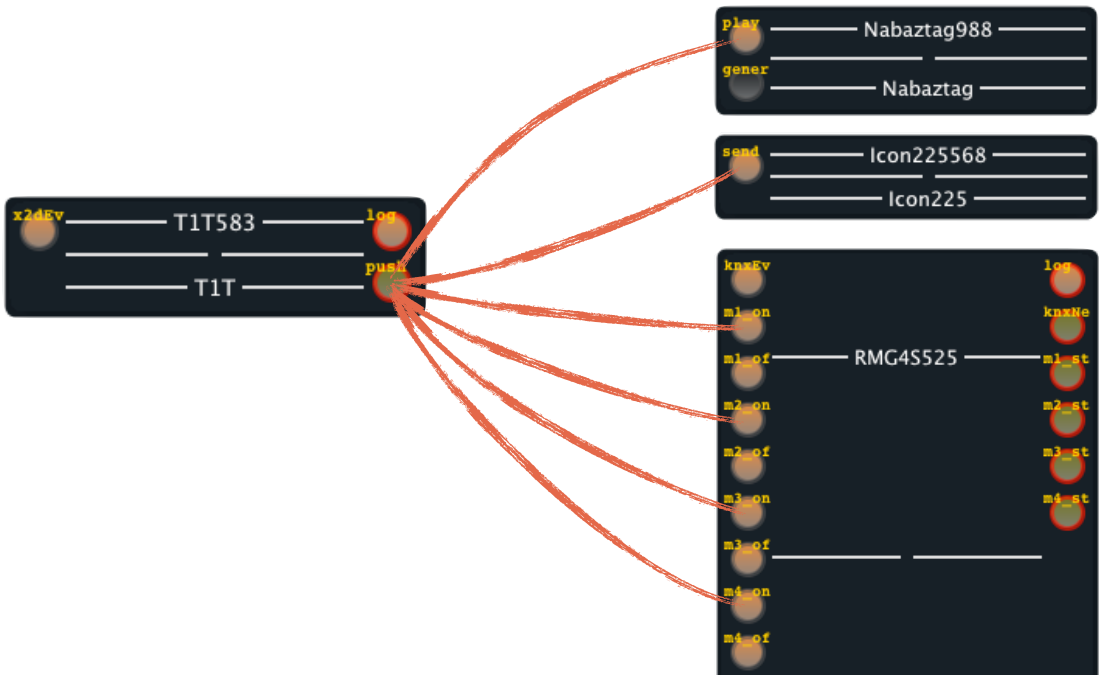


Figure 8.4: Components used in the evolution experiment

procedure is transparent for Mrs P., who is called just before and after the operation to keep her informed. The OSGi runtime prevents any service interruption.

8.5.3 Results

Again, a driver had to be created as well as a component to handle the RMG4s. Thanks to the Model@Runtime engine, the technician was able to collect the model and modify it. He then asked the runtime for the deployment of the new model. Realized through a TCP/IP connection, this evolution could have been realized remotely. This ability reduces the disturbance for the helped person, and can reduce the time from the query to the realization.
For this experiment, the communication with the runtime used a TCP/IP connection, which may not be that easy in real life.

8.6 Adaptation issue

The behavior of the lights was sub-optimal after the deployment of the evolution. Indeed, the lights were switched on, whatever the period of the day, every time the remote control was pushed. To remedy this issue, an adaptation mechanism (presented in [NFM+10]) was deployed. This mechanism changes the configuration of the system, according to the external lighting value sensed by a KNX outdoor weather station.

8.6.1 Test Environment

In addition to the previous equipment, selected for interoperability experiments and evolution concerns, the outdoor KNX weather station is now included in the configuration.

8.6.2 Resolution Protocol

The driver for KNX products already exists. The unique thing to implement is a component representative for the weather station.

The Model@Runtime engine offers means to connect reasoners. A reasoner is able to modify (or ask for the modification of) the running system configuration. The decision is made locally according to some contextual information. This information can be as simple as a value change, or as complex as an aggregation of events. For this evaluation, a reasoner has been created to modify the connections of the *RMG4S* according to the period of the day.

The system is currently composed of five components. A *TYXIA_110* remote control, a dispatcher connected to a 3G USB stick to send texts, and to a Nabaztag rabbit to inform Mrs P. In addition, the RMG4S makes it possible to control the lights. **At night**, this configuration is the one required. **During the day**, the RMG4S should not be activated, and the connections with the dispatcher have to be removed.

The weather station has been added to the system as described in section 8.5. Figure 8.5 presents a view of the system with the meteo station integrated.

To be able to perform execution time bench testing, the experiment including the reasoner was deployed from scratch. The MSI Top was cleaned of any previous binary element and restarted. The deployment was then realized as follows:

Initial Deployment The initial deployment is realized during the day. The model deploys only the remote control, the rabbit, the USB stick, the dispatcher, and the weather station. In this configuration, the elderly person can ask for help by pressing the Tyxia 110 button, just as before, but no light is switched on.

At night The reasoner adapts the system to the new conditions. It changes the model by adding an RMG4S instance and all necessary connections to the dispatcher.

Day When the day returns, the reasoner removes the connections and the instance of RMG4S.

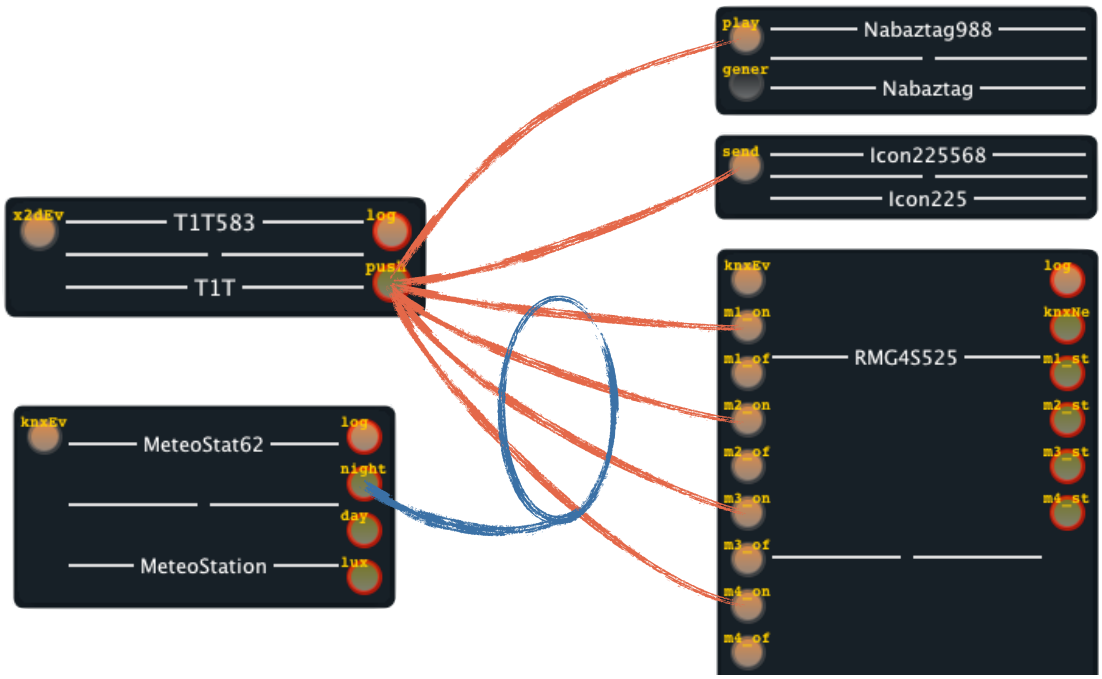


Figure 8.5: Components used in the adaptation experiment

8.6.3 Results

Figure 8.6 presents the execution times measured while a sequence of reconfigurations of the system was run. This sequence consisted of five steps. After the initial deployment (State 1), the scenario iterates night states (State 2 and State 4) and day states (State 3 and State 5) during the next two days.

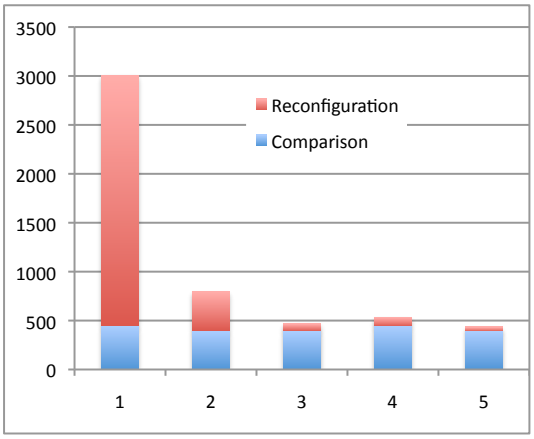


Figure 8.6: Time (in ms) spent in Configuration Comparison and Actual Reconfiguration

As the worst case scenario has been considered, the system is initially empty. Starting from scratch, all components need to be deployed during the initial configuration. In particular, all the component types have to be downloaded and checks have to be performed on the entire model. It explains the rather long reconfiguration time of step 1: 2.5 seconds.

The first reconfiguration (day → night, step 2) implied the deployment of an instance of RMG4S and the creation of bindings. As this component has never been used before, its component type is not present and has to be downloaded. All other components already deployed are reused. The downloading and deployment of the component type, plus the instance creation and its bindings to the dispatcher, are realized in less than 400 ms.

The next 3 reconfigurations (night → day → night) are much faster. Step 3 simply consists in unbinding and removing the RMG4S component. Step 4 is similar to step 2. Component types are not uninstalled. The RMG4S component type is thus immediately available for an instance creation. Step 5 is similar to step 4. The actual reconfiguration time of these steps is less than 100 ms.

For each reconfiguration, a model comparison is performed by the Model@Runtime engine, prior to the real deployment. This comparison detects changes and creates the commands' sequence for the transition. This model comparison takes an almost constant time of 400 ms. Executed before the actual reconfiguration, the comparison delays the reconfiguration of the system, but does not impact the duration of dynamic reconfiguration.

8.7 Openness issue

As presented in section 8.2, the system was required to expose the devices on both UPnP and DPWS networks. The next two sections describe how components have been automatically wrapped to be available on these networks. They report an actualized version of the work presented in [NDBJ08].

The aim of these wrappers is to make the devices installed on EnTiMid, available for third party applications or other control devices, through several application-level protocols. It is thus just a mean to share the access to some devices and may not provide all the functions of each device.
UPnP and DPWS could have been implemented as home automation network technologies rather than just wrapping protocols, which would have provided full access to devices' functionalities.

8.7.1 UPnP export

UPnP [upn] is based on a discovery-search mechanism. As a UPnP-Device joins the UPnP network, it sends an XML description file to all UPnP-ControlPoints. This file presents the device with information such as manufacturer, device type, device model, or its UUID. Most of times a UPnP-Device is self-contained.
It is able to describe itself and the services it publishes on the network. The description structure, visible on the upper part of figure 8.7, is organized as follows.
UPnP specifications allow devices to contain other devices (called embedded devices). In this case, the container (called the *rootDevice*) takes the responsibility for publishing information about itself and each device it embeds.
Each service a device can offer has to be described in a separated file. This file characterizes all the UPnP-Actions the service renders, and all the UPnP-State Variables used by these actions. UPnP-Actions can admit parameters. These parameters have a direction (in or out), a name, and a related StateVar. UPnP-StateVars handle information such as value types, or lists of allowed values for a parameter.

8.7.1.1 Test Environment

This experiment required some devices to be deployed on the runtime for them to be exported. We chose to fix the system in the night state of the previous experiment, thus with a maximum of devices present in the system.
This test also required a third party tool to act as a UPnP external control point. This was the touch screen provided by Mrs P.'s son. Since EnTiMid was deployed on the MSI Top, we made use of a toolkit from Intel: "Intel® Tools for UPnP Technologies (Build 2777)". This toolkit is no longer maintained and is only available for Windows.

8.7.1.2 Resolution Protocol

Mapping UPnP devices to EnTiMid devices
Although EnTiMid devices and UPnP devices are quite similar, they are not exactly aligned in their structures. However, the mapping (blue arrows in figure 8.7) was quite natural. EnTiMid devices were mapped on UPnP devices.
EnTiMid devices can provide two kinds of ports: service and messages ports. *NB: only input ports are considered here.* Services ports are composed of operations. This kind of ports was associated with its UPnP equivalent, namely Service for the port, and Action for the operations. The closest UPnP element to handle message ports from EnTiMid is the concept of *Actions*. Indeed, a message port provides only one service/action. As a consequence, there are as many services as there are message ports created. Each service proposes a single action, which connects to the message port.

Generation of description files
In EnTiMid, each component is described by a model. The model is a graph of objects at runtime and is serialized in an XML file. The generation of the XML description file

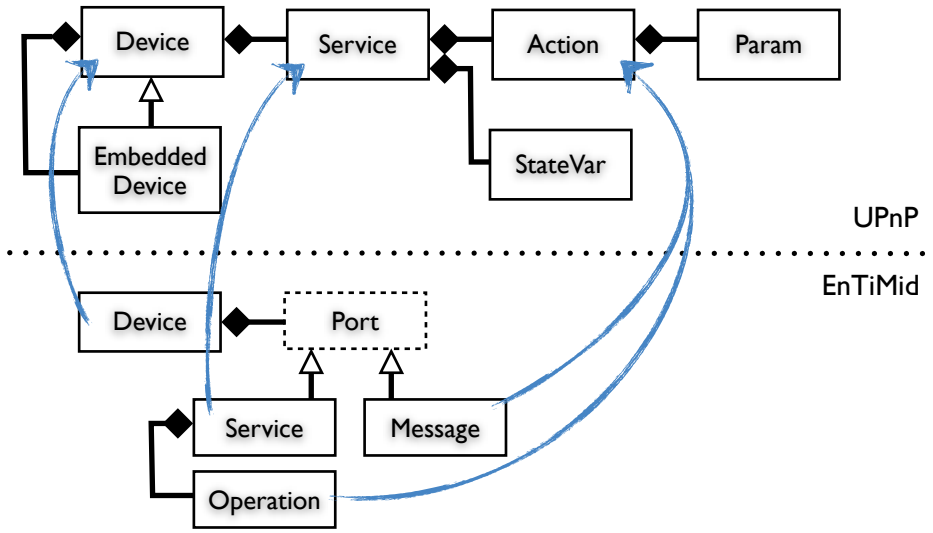


Figure 8.7: Mapping UPnP-EnTiMid

for UPnP is thus quite natural.
The service descriptions are the first files to be generated. For each EnTiMid service, a new file is created. Operations of the service are then described in the UPnP formalism. Not all operations are described, due to a sometimes complex translation between EnTiMid and UPnP runtimes. In any case, all simple methods are included in the description. In the same way, a new service file is created for each message port. The service takes the name of "<port>on<component>" for the services to be differentiated. Each service of this kind is composed of a unique action whose name is identical to the one of the port.
Afterwards, a separate description file is created for each component available on the system. The description of a device in UPnP is threefold. The first part provides general information about the product such as the name of the manufacturer, a brief description, the model type, the model number, or its serial id. If not available in the model, these elements are automatically generated, or completed with default values. The second part contains the list of the services a device provides. This description makes reference to the service description files previously generated. The last part contains information about the embedded devices. For each of them, the two previous parts have to be specified.

When a UPnP-ControlPoint starts, it broadcasts a query for descriptions of devices available on the network. The UPnP wrapper then sends the description files of all devices and services provided in response.

Runtime elements
The generation of description files is necessary, but not sufficient for UPnP queries to

be forwarded to the real device. Indeed, no connection was made between the real runtime component standing for the devices and the UPnP network. To cope with this issue, virtual abstract EnTiMid components are created at runtime. Each real device exported on the UPnP network is linked to an abstract component responsible for the handling of communications between the real device and the network requests. These abstract components only have output ports. *i.e.*: an output port is specially created on the abstract device to be connected to each input port of the real device exported on the UPnP network.
Then, queries are routed by the UPnP exporter to the abstract component in charge of the concerned device. The abstract component activates the input port of the real device according to the request.

8.7.1.3 Results

Once the wrapper is deployed on the MSI Top, the Nabaztag rabbit, the 3G USB stick and the RMG4S were all made available through the UPnP network. Indeed, we were able to view and act on these devices from a remote PC equipped with Windows, and the Intel UPnP toolkit.
The low number of tools that accept the discovery of self-describing UPnP devices can be a limitation for the use of this wrapper.

8.7.2 DPWS export

DPWS [JMS05] defines a minimal set of implementation constraints, to enable secure Web Service messaging, discovery, description and eventing on resource-constrained devices. Its objectives are similar to those of UPnP. The difference is that DPWS is fully aligned with Web Services technology and is designed to work upon a web-service transportation protocol. It also includes numerous extension points, to allow for seamless integration of device-provided services in enterprise-wide application scenarios.
From a conceptual point of view, the DPWS structure is close to that of UPnP, described in figure 8.7. Consequently, the mechanisms to map EnTiMid devices and their DPWS representative follow the same idea. Nevertheless, the generation process is different. Publications to the DPWS network have been realized thanks to the WS4D project [ZBB+07]. In their approach, each DPWS-compatible device has to extend an abstract DPWS device, proposed by the framework they provide. The reason is that this abstract component handles all web service-specific communication concerns. The creation of a virtual component is not sufficient in this case. A source code has to be generated.

8.7.2.1 Test Environment

Since UPnP and DPWS are very close in terms of needs for devices to be exported, the same set of devices was selected for this experiment.

As for UPnP, we made use of an external tool to check that the export of devices was made properly. The experiment was carried out using a second PC equipped with a DPWS explorer [3] to list and act on published devices.

8.7.2.2 Resolution Protocol

DPWS devices creation

For each device, service and operation, a Java class has to be generated. According to the element they represent, classes must extend *HostedService* for services, *HostingService* for devices and *Action* for operations. Parameters are instances of the class Parameter. Luckily, all these classes still can be generated with an automated process. To achieve the code creation, the JET Framework has been used. Templates of DPWS files were set up and they are used at runtime to produce Java classes.
More than just Java classes, the generated files are also implementations of new component types. These types are the wrappers of real devices for DPWS. These components are thus responsible for the direct connection between model elements and DPWS controllers.

Compilation and use

The generation process produces Java classes, but no binary code. These classes still have to be compiled to be useful at runtime. The decision was made to embed the JDT compiler provided by Eclipse. As a result, the bundle to export devices through DPWS is a bit heavy. The compilation is also resource-consuming for quite a small computer device. Once compiled, these classes are packaged into a bundle, which is then deployed on the OSGi runtime.
Classes are then handled just as classical components. The tool asks for new components to be added in the runtime, and they are bound to the device they export.

8.7.2.3 Results

This approach allowed to publish installed devices and act on them using the DPWS Explorer tool. Another approach involving external computing resources has been proposed to perform the class generation. The idea is to send a model of the devices to be generated, and get back a complete bundle containing the required component types. Altough this solution only requires to extract the mechanism, it has not been realized yet and is part of a future work.

3. http://ws4d.e-technik.uni-rostock.de/dpws-explorer/

8.8 Threats to validity

8.8.1 Internal threats

8.8.1.1 Variability management

Variability management, described as being an important issue in home automation for assisted living is not addressed in this experiment. Indeed, the scenario considers a unique deployment, for a single person, in a single home. A second round of definitions of a more global scenario may have stressed this requirement for variability management. Nevertheless, a lot of work has been carried out to try to cope with this issue, such as an approach using Aspect-Oriented Modeling presented in [MBNJ09]. Other perspectives to address this question are presented in section 10.1.4.

8.8.1.2 Scalability

The scenario did not highlight any issues about the distribution or scalability of the solution for deployment on a town-wide or even a countrywide scale. This scalability validation will be addressed in the future with real deployments.
The MSI Wind Top, on which the experiments have been led, may not be the unique platform on which to test EnTiMid and a large-scale vision scenario would have highlighted this.

8.8.1.3 Safety and Security

Voluntarily, we decided to distance ourselves from access security and privacy considerations. Not because they are not essential, but because they impose such heavy constraints that the search for a technical solution may have been compromised. Now that the system is clearly designed and that the proof of concept has been validated, work to secure communications and data has to be realized prior to any real deployment.

Concerning safety, our experimentations did not require complex checks on models. Only simple structural checks, to find cycles for instance, were implemented and used. In our approach, type checkers and validation policies have been designed to be customized according to the application domain and its constraints. Thus, their complete definition would have been useless in the context of the experiment. In the case of real deployments, they have to be completed to verify that no configuration identified as a case of failure is asked for deployment.

8.8.2 External threats

8.8.2.1 Validity of the scenario, real deployment

The experimentation scenario, even when defined in collaboration with several players in the AAL domain may not consider all cases. The interface between people coming

from a technical field and people coming from social field is quite difficult to find. Because people in social activities are not aware of what it is possible to do with technologies and because industrialists are not aware of the everyday problems the dependency of persons can raise, the discussions can rapidly come to a dead end.
The scenario validated for this study was accepted by all sides, but may be limited by the comprehension each side had of the problem.

EnTiMid has been instantiated and validated on a virtual case. A real deployment would have highlighted other issues, other constraints not addressed in this evaluation. This real deployment is part of the perspectives for this work.

8.8.2.2 Communications with smart devices

Gateways are essential for us to be able to communicate with smart devices. For instance, the bi-directional communication with Delta Dore devices was made possible by an R&D product. Otherwise, the *TYDOM 350*, an embedded web server, is the only device they commercialize to act on their devices. This one only enables users to act on devices through a web page interface. This product does not detect any event on the X2D network; it just acts on devices (with no acknowledgement by the way).
EnTiMid is not able to use any device without a means of communication with it.

8.9 Conclusion

An experimentation using real devices was set up by a collaborative work of the project's partners to evaluate EnTiMid on a scenario as realistic as possible. Made with the background of each partner, this scenario was designed to stress several issues identified on this kind of integration systems.
EnTiMid passed the main requirements highlighted by the scenario and required for such systems to be deployed. This validation comfort the idea of making EnTiMid an integration platforms to offer customized solutions for each person's needs.
Some limitations due to the lack of real and large scale deployment have been identified. These limitations will probably be addressed by the project of the company in charge of promoting this technology in the industry.

Part IV

Conclusion and Perspectives

The law of unintended consequences pushes us ceaselessly through the years, permitting no pause for perspective.
Richard Schickel

This last part wraps up the thesis. The first chapter summarizes by going back over the context and requirements, recalls the contribution and emphasizes its adequateness in relation to the requirements. After which, a short section discusses the benefits and limitations identified in this thesis.
The second chapter of this part shapes some perspectives of scientific development for this contribution, while the last chapter presents industrial perspectives.

Chapter 9

Conclusion

This chapter globally summarizes the work carried out for this thesis. Starting from the requirements, this chapter goes through the contribution, discusses its appropriateness to the context and ends by highlighting some benefits and drawbacks.

9.1 Reminder of Context

The ageing of the European population prompted the community to search for solutions to support this evolution. In this context, several issues have to be addressed in parallel. Firstly, the domain of health care suffers from a manpower shortage that could result in a decrease in the health service quality. Secondly, places in health care centers are not indefinitely extensive and centers will shortly reach their maximum capacities. Finally, a day spent in health care centers, or hospitals, is quite expensive and funding is limited.
Several projects have been started to try to address these issues. The Ambient Assisted Living (AAL) joint program has been created to promote such projects and emphasize the interest of Europe in advances in this domain. The *Innovation Domicile Autonomie (IDA)* project, initiated by the City of Rennes and the Greater Rennes Area, fits into this scheme with an evaluation of how the use of Information and Communication Technologies (ICT) can help to cope with these problems.
After a precise assessment of elderly people's needs, this project measured the adequateness of several industrial solutions to help and support elderly people at home. Among others, home automation technologies have been analyzed to work out their possible contribution to this problem. Rapidly, the survey demonstrated that a unique solution cannot be applied in all cases. Each person has different needs and requirements, which implies that the solutions need to be adapted for each deployment. Also, manufacturers reach their limits when a device has to be specialized for each user.

The technical solutions designed in this context require some software systems to bridge the gap between mass market home automation devices and customized solutions. To meet certain needs, these software systems must cope with several requirements.

9.2 Summary of requirements

Interoperability is the first requirement software systems have to cope with. Indeed, solutions proposed to improve elderly people's comfort at home may be composed of multiple products, from different manufacturers. Each device taking part in a solution addresses a particular need of the person and makes the solution closer to the ideal one. In any case, elements of the solution have to communicate with each other to render a global service, but the diversity of manufacturers makes the interoperability of devices a real problem.
The definition of a common communication interface for all components of the system could solve the problem, but it requires that all devices are re-engineered to implement this communication interface. With this approach, all products already available cannot be used, because they will never implement this interface. Since the solution must not be limited in terms of usable products, it avoids the definition of a global communication interface.

Adaptation and Evolution are the two main concerns to deal with in this domain. Software systems dealing with objects or services linked to actions of everyday life, have to take into account the environment in which they are being executed. They should be able to dynamically adapt to changes while running, in order to maintain a level of services, or functionalities. These adaptations should not require any restart of the system, since it would disable all functionalities for the time of restart. This is a real problem considering the trasmission of a request for assistance.
Needs, uses, protocols and technologies are changing. Some functionalities may finally be required, whereas others can become useless and need to be uninstalled. Security or communication protocols can be improved and deployed in new versions that have to be taken into account without needing to re-implement the entire system. Software systems must be ready to accept future and unforeseen evolutions, such as the installation of new services/functionalities.

Openness and Remote Control are intended to make all functionalities of the software system available for third party applications. Indeed, the connection of some products may require that the system is accessible through a specific application protocol. This availability offers means to connect to the system with no need to be aware of its internal organization. The only interest is on using the functionalities or devices. It is an open door for new unforeseen appliances and for external contributions. Each one can take part in adding a smart behavior on top of a reliable set of functionalities. Remote control may be required for a carer to remotely check if everything is doing well. It may also help in supporting remote-assisted management of the home for instance, or to remotely deploy new appliances or maintain the system.

Variability Management and Distribution issues are linked to the need for customization of solutions, and to the dispersion of deployments. Since solutions for elderly people have to be customized to best fit their needs, the set of options of the

system may become barely manageable. Some deployments may require high computation power, whereas others may run on tiny execution platforms. Moreover, evolutions of services or protocols may not be applied to all running systems at the same time, making the management of versions even more complex. All these elements lead to a huge number of configurations and to complex variability management.
Software systems have to be aware of these problems and offer tools in order to help system designers and technicians. Decision-helping tools should be created to support the design of software systems based on requirements and available devices.

Safety & Security is a very important concern for home automation systems. It is even more important when the system is aimed at improving the quality of life of dependent people. A minimum service level has to be guaranteed, for inhabitants not to remain stuck in the house in the case of an emergency for instance. Moreover, access to the system has to be secured to disallow anonymous control, but not tedious for authorized people in a normal use case.

Acceptability & Accessibility are issues that must be addressed, particularly when a software system takes responsibility for part of the home management. The AAL domain is a complex environment in which solutions must support the activity of elderly people and help carers in their work. For both of them, the system must not be perceived as a new constraint, or considered as stigmatizing. People must accept the solution deployed in their home and have to be reassured that they will keep a hand on things that could happen.

9.3 Survey of existing approaches

Among all the requirements listed in section 9.2, the survey of existing approaches concentrated on interoperability, adaptation, evolution, openness and variability management.

Scientific literature abounds with proposals using different approaches to cope with interoperability, adaptation or remote control concerns in several applications. Generally, service-based propositions [All11, **?**] sound helpful in targeting the interoperability of devices, but clearly lack description of the running application once deployed. They also bring essential ideas to properly handle the sporadic appearance of elements, since a service can be started and stopped at any time.
Component-based architectures [GMK02, RvdLKM00, BCL$^+$06] provide an ideal abstraction level to meet the requirements for a virtual representation of home automation devices. However, components' ports are often defined by an API. This strict definition may disallow some connection unforeseen at design time, without the help of *ad-hoc* connectors.
Using components for SOA [sca, MRRS10, EHL07] is certainly the best approach for

our concerns, since the benefits of the first balance out the drawbacks of the second. Transversally to any approach, model-driven engineering methods and techniques come with a lot of tools for virtual element manipulations. They seem handy for runtime management of devices, for the description of software systems, and for variability management.

All the approaches, tools, and frameworks considered in this survey have been reported in table 9.1. This table synthesizes the strengths and weaknesses of each approach in relation to the requirements identified.

9.4 Outline of the contribution

Inspired by achievements in electronics this thesis contributes to improving the flexibility of software systems, while maintaining a high level of reliability. The contribution is threefold.

(1) A new component model, which improves flexibility to enable the connection of heterogenous components.

(2) Tools from model-driven engineering, to create, edit, simulate and validate the structure and behavior of component assemblies, prior to their (re-)deployment.

(3) A runtime environment built on top of a service-based architecture, to support evolutions, adaptations and openness required by the proposed component model.

The implementation of this contribution called EnTiMid is composed of several elements. Each of them, presented as a layer, addresses a particular concern of the global problem.

Device Interoperability takes responsibility for the communication with real devices and between their virtual representatives in the *Component Model* layer. A mix of drivers (to connect the real to the virtual world), and asynchronous message-based communications, enables the connection of components previously marked as not compatible.

The **Component Model** layer brings up the necessary structures and methods to handle the virtual representation of real devices. It provides a unified description of possible actions and available information, using the paradigm of ports. In this model, ports can be of two kinds: synchronous (service ports) or asynchronous (message ports). This component model helps to provide a detailed view of components, with precise information for the *Model@Runtime* layer to work properly. Tools have been made available to ensure the synchronization of models and implementation codes of components.

The **Model@Runtime & Checkers** layer involves necessary tools to ease the management of the system. Specificities of components' implementations are invisible

at this level, thanks to the *Component Model* layer. Simulations and checks can be safely performed at this level of abstraction, with no consequences on the running application, since the model view is synchronized but independent from the execution. *Model@Runtime & Checkers* contribute to enabling management of the system at runtime, to offering tools for checks and validations, to improving the safety of the solution, and help in dealing with the variability of the system.

The **Wrappers** layer takes responsibility for publishing the devices present in the system, on application level networks. This ability opens our solution to existing and future protocols. Often too heavy to be embedded in basic devices, this layer makes all devices available on application level protocols for free.

Service-Oriented Runtime comes to completes the contribution by providing an execution environment for the new component model. It brings life to the *Model@Runtime* by supporting dynamic *adaptations* and *evolutions* while running.

9.5 Adequateness of the contribution

The AAL context, the home automation domain description and the state of the art, led to an extraction of a list of requirements. These requirements have been stressed as the essential abilities a software system should be able to provide to be used in this context. The table 9.1 recalls the table presented in chapter 4, in which all existing approaches were presented, and where their participation with relation to each identified concern had been reported.
The last line completes this table with EnTiMid, and shows that it fits most of these requirements.

The Device Interoperability layer, helped by the Component Model, addresses the interoperability requirement. Openness is ensured by wrappers, for application level protocols, and by drivers for future manufacturers. Adaptation at runtime and evolutions are made available by the use of Model@Runtime techniques and the OSGi runtime for supporting the implementation of this contribution. Variability management is simplified by the presence of a model, but tools are still insufficient to properly cope with this issue. Finally, remote control is possible thanks to 1) the Model@Runtime for remote adaptations or evolutions, 2) the wrappers for remote application level control.

9.6 Conservativeness

During the state-of-the-art survey, some good properties were identified. The contribution of this thesis is moderate in relation to these properties.

		Interop.	Openness	Dynamic Adaptation	Static Evolution	Variability Management	Safety & Security
Generic Approaches	OSGi [All11]		+	+	+		
	ESB [Cha04]	+	+		+		
	Darwin [GMK02]			+	+		+
	Koala [RvdLKM00]				+	+	+
	Fractal [BCL+06]			+			
	SCA [sca]		+		+		+
	FraSCAti [MRRS10]	+	+	+	+		+
	iPOJO [EHL07]		+	+	+		
Domain-Specific Approaches	uMiddle [NT07]						
	SOPRANO [WSO+10]			+	+		
	Gaïa [RHC+02]	+		+	+		
	Dia Suite [CBC10]	+	+			+	+
	Habitation [JRS+09]	+				+	
	WADL [CDT08]	+		+	+		
	PervML [MPC06]	+	+	+	+	+	
	AutoHome [BDLM11]		+	+	+		
	WComp [FHL+11]	+		+	+		
	Niagara [Tri08]	+	+				
	EnTiMid	+	+	+	+		

Table 9.1: Adequateness of the contribution

The **Reflexive Model** proposed by MDE is still available. The Model@Runtime layer is responsible for this ability and keeps an explicit and independent model reflecting the architecture living at runtime synchronized. This model allows for the creation of reasoners, able to perform changes on the model, with no immediate impact on the running system, until the model reaches conformity.

Externalized coupling is provided by the Model@Runtime by enforcing the elicitation of links between components. Moreover, the presence of drivers imposes the components to be created with no idea about their usage. This externalized coupling makes it possible for reasoners to dynamically change component connections, and even components directly. The enforcement of the component independence requirement, to allow interoperability, also takes part in ensuring the externalization from the component implementation.

Hot deployment is natively supported by the OSGi runtime execution environment used for EnTiMid. Indeed, for reasoners to be able to adapt the system with new components, or for evolutions to be easily deployed, EnTiMid had to preserve this property in the final solution.

Close Isolation enforcement is imposed by the device interoperability layer and the entire EnTiMid system. Types and instances are handled separately and all instances have independent life cycles since real devices can evolve independently.

Openness was identified as a good property, but also as a key requirement, which has been addressed in this contribution, and explained in the validation.

9.7 Immediate benefits

9.7.1 Development of components made easier

The tools coming with this contribution aims at supporting the development of components. The component model forces developers to respect properties such as close isolation, or externalization of component couplings and makes the maintenance and the evolutions easier. The use of annotations in the component code and the availability of a code generator also simplify the everyday work of component developers. The code generator and the synchronization mechanism bring significant gains in terms of prevention of errors, time consumption and shorten the time elapsed from a new requirement to the solution.

9.7.2 Simple creation of applications

Thanks to the component model and modeling tools, the creation of applications is made very simple. Libraries of components can be imported into editors and components are assembled and connected using drag-and-drop interactions. If checkers so authorize, connections from port to port make it possible to connect any port to any other, whatever their roles or actions.
Designers of home automation devices are already familiar with this way of connecting things, since the model is very close to that of the electronic components. The design of a solution customized to meet a person's specific needs is thus quite quick and simple for engineers and technicians.

9.7.3 Sustainability and precision

The adaptation and evolution abilities of EnTiMid improve the sustainability of deployed solutions. Present by default in the system, they offer the support for the evolution of both technologies and elderly people's pathologies, with no need to change the entire system. This way, a software system created to meet specific needs at a time, can be changed with a very limited cost, which makes the solution always precise and sustainable.

9.7.4 Seamless integration of IoT and IoS

The new component model enables the connection of heterogeneous components not designed to be working together. The heterogeneity can be due to the difference of man-

ufacturer, protocol used, or media used, but can also be due to the object they represent. Several services available through the Internet have been wrapped into components, to enable the application to access a service such as an on-line calendar, a weather service, a picture-sharing service and even a famous social networking service. The component model enables seamless connection of a Google Calendar from the Internet to a light component for instance. The effect of such a connection could be to switch off the light when no appointments are specified.

9.8 Limitations identified

9.8.1 Behavioral description

The component model eases the structural description of a software system when people are very familiar with the description of its behavior. In addition to the component model (thus the structural description), it would be helpful to have a second tool to check and describe the behavior of the system. In this condition, an end user could be able to change the behavior of the system, without dealing with the structural description.
If the answer has not been provided yet, it may be because the problem is not simple once the behavior of the system can be described in several pieces. Indeed, thinking in a functional way makes people define how the system must behave when the door opens, or when an alarm is triggered, but with no consideration about the consequences on its global behavior. Moreover, the structural and the behavioral descriptions have interactions with each other.
Lastly, as non-experts of the domain, the description people can make of how the system must behave never takes errors or failures into account. From a linear description, erroneous paths have to be guessed and tested [OP97].

9.8.2 Port parameters

Classical component models have been excluded in this thesis, because of the too strict specification of ports, making it impossible to connect two ports if their APIs are not aligned. But the problem of alignment has not completely disappeared in our component model. It has been moved from the implementation to the virtual representative of each component. Thus, the alignment of parameters passed through the ports has to be resolved at model level, before deployment.
Although this problem has not been addressed in this thesis, it has already been identified and scientists have already proposed some solutions under terms such as component connectors, which can have complex behaviors (cf Beugnard et al. in [MB05]). Mechanisms such as renaming or mappings [CBJ10] of parameters could be implemented to cope with this issue.

9.8.3 Too weak checkers

Our experimentations did not require complex checks on models. Only simple structural checks, such as cycle detections, were implemented and used. Many checkers had not been completed since they were related to the business targeted. In the case of real deployments, they have to be completed to verify that no configuration marked as failing is asked for deployment.
Used at different steps, the checkers are different and they address various aspects of the application. Thus, the information, required to be able to perform each check properly, depends on what has to been checked. Since no complete checker had been implemented, the information currently available in the model could be too poor for checkers to work.

9.8.4 Variability management

Variability issues have not been completely addressed, since a small set of components is sufficient for testing. Also, the number of components in the IDA experimentation was small enough to be handled manually. In the perspective of real deployments, the variations of configuration will impose the creation of tools to help in facing this huge variability.

9.8.5 Improvements for embedded platforms

In the context of the IDA project, the choice has been made to deploy EnTiMid on a touch screen PC. This PC has high computational power and lots of memory compared to some more embedded platforms. But the deployment of a touch screen PC may not be necessary in all cases and some more embedded devices may be sufficient to automate some tasks. In any case, the runtime of EnTiMid requires that a Java virtual machine is deployed on the platform first and thus providing enough power for the JVM to run.

9.9 Contribution to the S-Cube NoE

9.9.1 The S-Cube Network of Excellence

S-Cube[1] is a European Network of Excellence (NoE) in Software, Services and Systems (S^3). This NoE aims at making European research the leader in the software-services revolution. By connecting research to industry, and unifying multidisciplinary researches, S-Cube aims to develop agile and holistic service engineering methods, and to specify principles and techniques of service adaptation.
This European NoE has been funded by the European FP7 'Coordination' Research Programme under the ICT theme. Along with strong collaboration and mobility opportunities

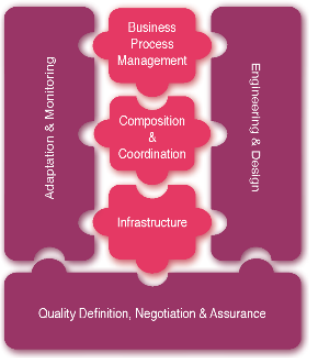


Figure 9.1: S-Cube Research Framework

1. http://www.s-cube-network.eu/

beyond European research centers, S-Cube has funded several PhD theses in different layers of the S-Cube "BigPicture" (fig. 9.1).

The skills of excellence of the INRIA Triskell team in which this theses was conducted, are dedicated to ease, and improve, software development methods, by the use of components, services, models and validations. Thanks to this team orientation, Triskell is involved in the S-Cube NoE, which has brought funding for two PhD theses. The research leading to the results presented in this theses has received credits from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

9.9.2 Contribution

The contribution of this thesis is aligned with the Work Package 1.2 : *Adaptation and Monitoring Principles, Techniques and Methodologies for Service-based Systems* of the Joint Research Activity(JRA) 1 : *Engineering and Adaptation Methodologies for Service-based Systems*

The general objective of the JRA-1, is to "devise an integrated set of principles, techniques and methodologies for engineering, adapting and monitoring hybrid service-based applications, while guaranteeing end-to-end quality provision and SLA conformance", according to the S-Cube description of work [2].
This thesis provides a new component model that: implies new engineering techniques and methodology, enables the adaptation of hybrid service-based application and offers means to perform checks and verifications to ensure the quality of services.

More precisely, the contribution of this thesis takes part in the JRA-1.2 work package, which aims to define novel principles and techniques for cross-layer monitoring and adaptation of Service-based Applications. If EnTiMid does not address monitoring issues, it actually copes with adaptation requirements.

From the S-Cube perspective, EnTiMid can be considered for handling adaptations of the infrastructure, or of the composition and coordination layer (see figure 1.1). Coupled with other layers, it can take part in the cross-layer adaptation mechanism.

2. DoW Amendment 4, December 6th, 2010

Chapter 10

Perspectives

10.1 In research

The contributions of this thesis leave some questions open, and have opened some doors. Therefore, the perspectives of this work aim at investigating and answering questions that have not been yet addressed, or go one step beyond into new uses of this contribution.

10.1.1 IDA, second phase

Conclusions from the experimentation led in the context of the IDA project shaped a promising future for the use of such tools, in the AAL domain. Even if the maturity of EnTiMid and the objective of the first phase of the project did not allow testing of EnTiMid in a real deployment situation, the protagonists (industrialists, carers, social workers and elderly people) have shown an interest in the provision of customized solutions for each person.
Currently, the second phase if the IDA project is being set up. Hopefully, it will be the moment for EnTiMid to perform the last checks and to validate both the scalability and variability management, on real deployment.

10.1.2 End User Programming

End User Programming [KAB+11] relates to the ability for anybody to program something. For instance, when a user programs the hours of start and stop of the heating system, he is actually programming. In the context of the IDA project and following the idea that inhabitants must be able to keep control of things in their homes, end user programming sounds like a very promising, but yet challenging perspective.

10.1.2.1 Which description language ?

Software developers like to be able to use the keyboard only. A graphical user interface, with drag-and-drop interactions to create assemblies, will probably not meet the

requirements of this kind of population. For them, a textual language seems to be the simplest thing.
On the other hand, inhabitants do not all have skills in programming languages, and especially not elderly people. They would probably express their requirements for the behavior of the system in another way. The question is which one.

We had a range of description tools at our disposal, from a textual domain-specific language to a visual language composed of icons and boxes, linked with arrows. A solution for this problem probably lies somewhere in between these two extreme proposals and is surely not unique. Indeed, for the same system, an elderly person will probably be lost in a textual language and an engineer may be frustrated at being unable to express himself as usual.

The validity of the behavior described is also challenging. End users may not have a global vision of the system and thus may ask for a behavior that could lead the system to failure. Secondly, people naturally express the nominal behavior, without being concerned about possible deviations of this behavior. To address this problem, tools have to be proposed to check the validity of the nominal behavior and track and check any possible variation in the scenario.

The unique and universal language for describing how things have to behave will probably never exist. Because each user has a different kinship with technologies, systems should offer several edition tools, out of which the end user selects the handiest for him.

10.1.2.2 Fuzzy Logic and Learning Algorithms

In the hypothesis that people are able to describe a behavior of a function, how could they know about the limits of values? In other words, if a user is defining a behavior for the light, how could he know the minimum and maximum values that the light sensor can sense ?
The fuzzy logic paradigm [CBBJ08, Men01] proposes to use terms and non-fixed values in decision algorithms. Indeed, a fixed value is never appropriate because a regulation value must be modifiable. This paradigm makes it possible to work with terms and rough values only, because thresholds are computed at runtime. During the execution, users can act on these thresholds by telling the system about good or bad situations. Quite close to ideas of artificial intelligence, this approach could be coupled with some learning mechanisms, to go a bit further and re-simplify description of an application behavior.

10.1.3 Distribution and Pervasiveness

The distribution of applications brings several interesting facilities, such as load balancing, or redundancy to cope with failures of system elements. This question has not

been properly addressed in this thesis, but may rapidly become a limitation. Moreover, working with devices brings EnTiMid close to the ideas of pervasive computing. In this domain, objects' interactions are controlled by invisible nested software systems. Invisible for users, these systems have to self-reconfigure to take into account changes in their environments.

In the perspective of a large-scale deployment, distribution and pervasiveness can both come out as key requirements for some deployments. In [DDNDM07], Devescovi et al. propose algorithms for the self-organization of autonomic systems using the SelfLet approach. According to the presentation web page [1], a SelfLet is a "self-sufficient piece of software which is situated in some kind of logical or physical network, where it can interact and communicate with other SelfLets". This definition is very close to the definition of a smart device and SelfLets could be included in devices and device-controllers, such as firmware, to ease their integration.
This approach could foster the distribution of EnTiMid on several nodes, help to prevent system failures, balance the load of resource-consuming components, or ease the connection of smart devices.

10.1.4 Architecture Synthesis

The architecture synthesis goal is to assist in the creation of an application. Feature diagrams and automatic derivations into products, templates and wizards guiding the developer through the steps of product design, are two examples of tools enabling the synthesis of architectures.

10.1.4.1 Dynamic Software Product Lines for the management of variability

Not really addressed in the contribution, nor experimented in the validation, the management of variability in the domain of AAL and Home Automation is still a real problem. Luckily, the omnipresence of the model in all steps of the application-making process enables the use of well-known modeling tools to help in handling variability.

As proposed in [Mor10], Aspect-Oriented Programming, coupled with Software Product Lines can be used to address this problem. Product Lines, a well-known variability management tool for supply chains, has been transposed into the software domain under the name of Software Product Line (SPL). Large scale productions such as that of cars handles the variability of customers' requests using these product lines.
A product line consists of a base product that can be augmented with options selected by the customer. Software systems with a huge number of variable elements, such as component-based applications, can be defined the in the same way. The base functionalities of the software are described in the base product and specific options are plugged in according to the customer selection. The problem is that these tools have been set

1. http://selflet.sourceforge.net/

up to ease the one-shot creation of a product.

Dynamically-adaptive software systems are able to dynamically evolve after their creation, and SPLs are no longer sufficient to help in handling the description of things that can be changed at runtime. To cope with this issue, Dynamic Software Product Line (DSPL) have been proposed. They enable the description of variation points during the execution of an application and make it possible to identify the exchangeable elements. The work carried out by Carlos Cetina et al. in this domain, presented in [CGFP09], is very close to what we want to achieve and reflects our future work.

10.1.4.2 How can the behavior be descibed?

In EnTiMid, mapping components to leaf features in the DSPL makes it very simple to describe the desired configuration of the software at a high level of abstraction. Nevertheless, components in EnTiMid are developed with a strong effort to respect the close entity principle and they do not know, or depend on each other. As a consequence, the DSPL can only support the description of the number of components, their types and the different options in the case of reconfiguration; in short, the structure of the assembly. Nowhere can the interactions between components be specified.

While still in its infancy, we proposed in [INPJ09] an approach combining DSPL and Business Process Modeling. It enables the description of both architecture and behavior, by a combination of two modeling tools. Once coupled, these two models describe the structure of the application and its required behavior, which makes it possible to generate the entire system with connected components. As work in progress, this approach still has to be experimented in more depth.
Cassou et al. recently presented another approach to this problem of describing interactions. In [CBCL11], they introduce a "behavioral contract". These contracts are aimed at offering means to express the set of allowed interactions between components and describe both data and control-flow constraints. The integration of this idea with EnTiMid may be studied in future work.

10.1.5 Kevoree

EnTiMid, as an achievement of this thesis, has been highlighted as an interesting approach to address some identical issues in other domains.

For the principles of this contribution to be used in other contexts, EnTiMid has recently been re-designed to become the customization of a more generic tool, specialized for home automation and AAL. Its name is Kevoree [2]. The core mechanisms of adaptation, evolution, etc. have been moved into Kevoree in order to make them available

2. http://kevoree.org

for use cases other than home automation.

Whereas EnTiMid is responsible for providing a set of services and components for Home Automation and AAL, Kevoree offers a set of tools for the component model. These include a framework to ease the implementation of components, a graphical editor to create component assemblies and a specialized runtime.
The development of Kevoree is actually part of the work in progress in the context of another thesis, which explores different improvements. For instance, questions about distribution and meanings of links between components.

Kermeta is a DSL optimized for metamodeling engineering. Developed in the TRISKELL team, it provides an integrated environment for Model-Driven engineering activities. Initially developed as a set of plugins for Eclipse, a work in progress is trying to make it run using the Kevoree tools.

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments, according to the manufacturer's web page. Recent developments shaped the idea of using Kevoree and the component model to ease sketches and the deployment of applications on Arduino platforms.

10.1.6 Open Control/Command Operating System

The component model of this contribution has been designed to allow for the connection of heterogeneous components. Inspired by electronics, the elements to be modeled and connected just need to be expressed in terms of components with inputs and outputs. Since almost all automated systems can be expressed this way, almost all systems can be modeled using the component model proposed in this contribution.
The independence of the model in relation to real devices-specificities enables the description of any application/system. Thus, EnTiMid could take on the role of a universal control/command operating system, since the only difficult point is to develop the driver in charge of the interface between the real world and the component level. If a driver for an application can be created, EnTiMid can help in controlling it.

10.2 In industry

The contribution of this thesis interests several audiences. The general public is curious to know about how computer science can help in improving elderly people's quality of life at home, since EnTiMid was initially designed for the Ambient Assisted Living context. Everyone has or has had a family member who could have been helped by such a proposal. Industrialists are more interested in the good properties of the contribution. Since device manufacturers are familiar with electronic components, they easily understand how this component model works.



Figure 10.1: Science festival presentation

Starting from the validation scenario, in just 3 years a demonstration of EnTiMid has been set up and presented in more than 10 public and scientific events. This demonstration stressed the adaptation aspect of the contribution in an AAL context.

10.2.1 Public events

Public events are an opportunity for the general public to discover what issues scientists are trying to resolve and how they do it. On the other hand, these meetings are also an opportunity for scientists to collect feedback on their work, from uninitiated persons. Uninitiated, in the sense that they are thinking about the problem for the first time. The questions asked are often very interesting, prompting you to step back from the details and to consider the proposal from a more global view.

The most important presentation was probably the science festival held in 2009 in Rennes, where EnTiMid had been selected to represent the INRIA laboratory. The science festival, *Fêtes de la Science*[3] in French, is a national event lasting for 3 days. From Friday to Sunday inclusive, scientists present their everyday work, explain the problems solved, or the phenomenon involved in some experiments.

During the first day, the festival admits only primary and middle school children.

3. See the videos (in French) at http://videos.rennes.inria.fr/fete-sciences-2009/index.html

Visits are organized by groups and presenters have to explain their work to people aged from 7 to 15 years. This day is the most difficult day of the all festival. Not because the questions are complicated to answer, but because the explanation must be understood by everyone. This first day did not bring a lot of useful comments, or in any case, less than the two other days.

On Saturday and Sunday, the festival is open, for free, to families and anybody interested in sciences. These two days were a hard test for EnTiMid and full of interesting discussions with people. Indeed, EnTiMid had to run from 8am to 8pm without failure, disregarding touch-screen stress due to children's fingers, and in spite of the unpredicted use cases requested live by visitors.
It was the place for EnTiMid to become enriched by new ideas, remarks, or people's experiences in facing elderly people's dependency problems. For the entire duration of the festival, no once did somebody say that EnTiMid was useless, meaningless, or that the use case was not realistic. "It is not always that simple in real life" is the only remark we got.

Open days of the University of Rennes 1, or the opening celebration of the ESIR, the newly created engineering school of the University of Rennes 1, have also been two moments for exchanges with a large audience. Concerned by the studies, visitors' questions during these two events were more precise and more focused on what had been realized and how.
These demonstrations were sources of very interesting ideas to improve EnTiMid.

Public presentations are very good opportunities to sense how the contribution exposed is perceived by anonymous people. None of them had a negative vision of the work, some had doubts and others asked about how to buy it. In my opinion it validates the utility of this contribution as perceived by people.

EnTiMid has also been used as support for a demonstration called "*Leveraging Models From Design-time to Runtime. A Live Demo*" described in[MNBJ09].

10.2.2 Industrial perspectives

The success of EnTiMid has generated some industrial contacts. Several companies were interested in the abilities of this software to adapt and evolve at runtime. Most of them, close to the home automation domain, saw in EnTiMid a great opportunity to enrich their products. But EnTiMid was just a proof of concept, a prototype of research not ready to be deployed in the industry. This is partly the reason why EnTiMid was not selected to be deployed in homes, in the IDA project.

Aware of this problem, a project of developing EnTiMid, to make it ready for use in the industry had been submitted to the Regional Council of Brittany and accepted. This project funded the work of an engineer for one year, whose task was twofold. Firstly,

to redevelop some parts of the contribution for it to be more stable, and secondly, to support the promotion of EnTiMid in the industry. The development methods of EnTiMid have been clarified and a second demonstration has been set up to focus more on the core functionalities and less on the home automation/AAL aspect.

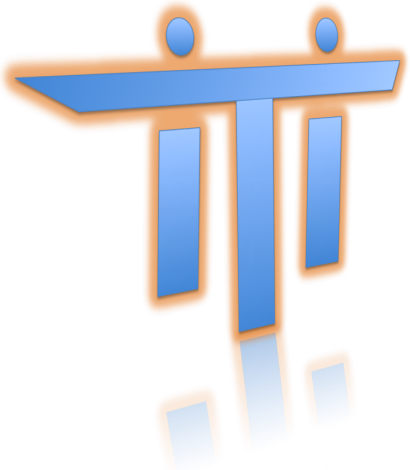
In addition, a project of company creation is currently in progress and should end with the creation of a structure in charge of the promotion and support of EnTiMid in the next few months. It was also a part of the development engineer's work to support the creation of this spin-off.

Part V

Appendix

Appendix A

ITI Project



The ITI(Intuitive Touch-screen Interface) project was a collaborative project involving the INRIA, and the LOUSTIC laboratory, in the global context of the IDA project. The LOUSTIC [1] is a laboratory of observation of the use of Information and Communication Technologies(ICT), which measures people reactions with relation to a given technology. Tests of artefacts are realized by volunteers, belonging to a pre-defined part of the population. According to the product under test, several measurements can be performed such as time spent to find an information, eye-tracking, reactivity of the product, etc.

EnTiMid is designed to allow dynamic and unpredicted changes in terms of components or services. Any end-user oriented control interface for an EnTiMid running system have to be able to deal with these changes at runtime. In this context, the creation of a universal Graphical User Interface(GUI) sounds like a complicated task. To cope with this problem, the idea of an automatic generation of Graphical User Interface (GUI) has been proposed. From this perspective, devices have been imagined able to provide an abstract description of the graphical controls they require.
In addition, GUI can be constraint by users' preferences, or needs in terms of font size for a an elderly person, or a nurse. The idea is to adapt the graphical user interface during the execution. Some work, presented in[BBB+11], has already been engaged in this way.

If the automatic generation of GUI appeared as a really interesting field of research, the amount of work to achieve a first proof of concept appeared to be huge. Therefore, it has been decided to first sketch an application GUI, adapted to elderly people, before actually performing the generative work. The sketch of this GUI, and the measurement of its usability by elderly people was the goal of the ITI project. It also validates that the GUI of this contribution is acceptable, and accessible, for elderly people, which are part of requirements listed in section 2.3.

1. http://www.loustic.net/

The results presented in this chapter have been extracted from the internship report [CC09] of E. Colas and N. Courtais, who actually realised the tests.

A.1 Presentation and Goals of the project

In the AAL context, people must be able to interact with the assisting system. Since a great part of the end-users in this context are elderly people, the control interface of any solution built with EnTiMid must be adapted. The adaptation of an interface to the elderly population can concern some ergonomic aspects, but questions about graphisms were treated as a secondary goal. The main concern was about the navigation method to be implemented. In the domain of home automation control interface, two methods of navigation can be applied. The first consists in selecting the location first, then the function to be considered. On the other hand, the function is selected first and the zone concerned after. Thus, the leading question of the project was: Is it more convenient for people to navigate by Zone then Function, or by Function then Zone ?
To answer this question, and design a relevant GUI sketch, the project had been split into three phases.

A.1.1 Phase 1

This phase did consist in designing the graphical interface, according to well know ergonomics rules [BS93]. In this phase, attention was paid to the size of each element (buttons, text, labels, etc.), for them to be easily readable, and for the information not to be lost in useless decorative elements. At the end of this phase, two graphical interfaces were released. The first was implementing the Function/Zone navigation mode; the second, the Zone/Function one.
Figure A.1 presents the graphical user interfaces created in this phase. Three screenshot of the Function/Zone proposition are visible on the first line of the figure, while the bottom left screenshot displays one of the three Zone/Function screen. All the widgets used are shown on the bottom right part of the figure.

A.1.2 Phase 2

Phase 2 aimed at presenting the interfaces to elderly people. Measures have been realized on their first reactions with relation to the navigation mode they preferred, and on their ability to use the controls and retrieve information from the system. The second phase improved the graphical interfaces in terms of controls, and information presentation, for these elements to be of minimum influence on the answers to the navigation question.
The main improvements concerned the widgets themselves. The light widget has been reduced to a single light bulb, which aspect clearly indicates the state of the light, and a touch of the bulb toggle the state of the light. This improvement is visible at the bottom

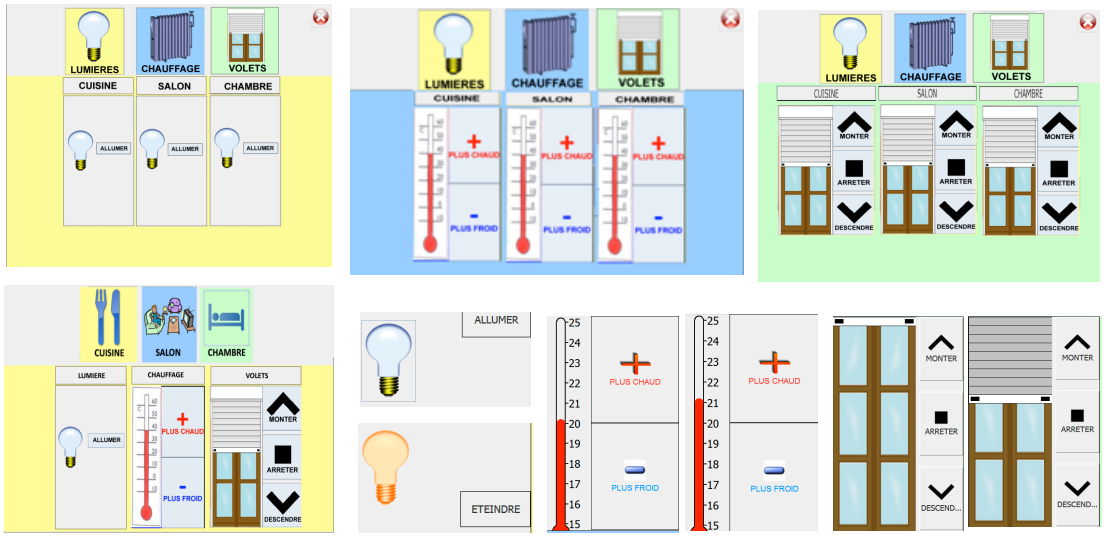


Figure A.1: Results of the first phase

of figure A.2. No change has been made on shutter widgets, but the temperature widget was simplified, by the removal of the plus and minus buttons, since elderly people just pressed on the thermometer.
The figure only shows Function/Zone interfaces, but the modifications on widgets have also been realized on the Zone/Function interface the same way.

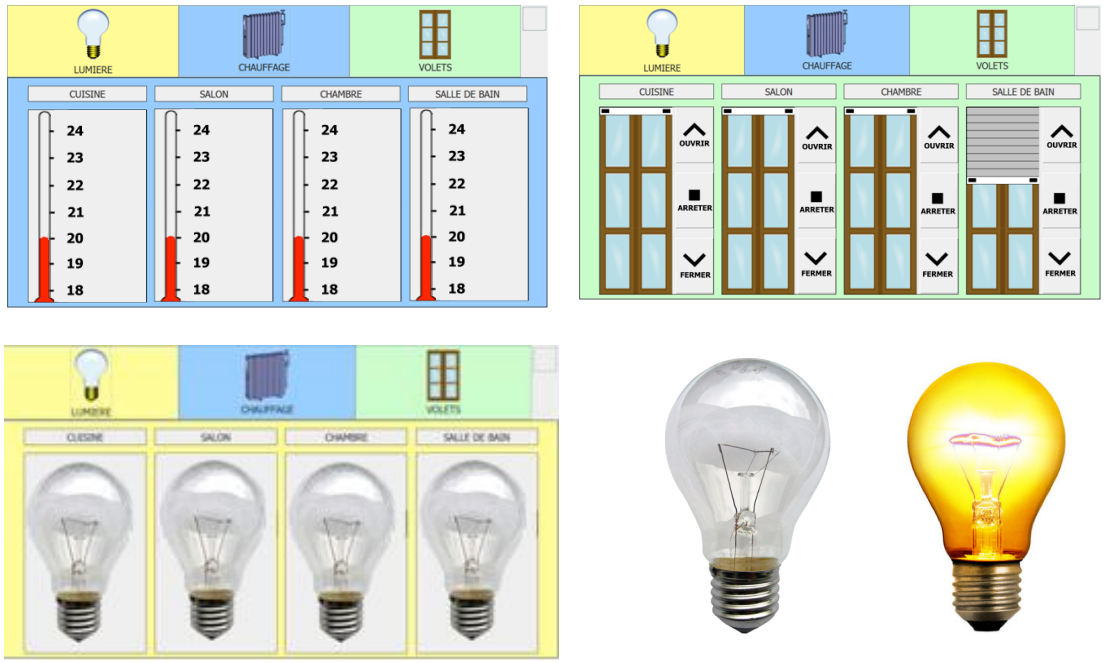


Figure A.2: Improved interfaces, result of the second phase

A.1.3 Phase 3

This last phase resumed the tests with elderly people, using the improved user interfaces.

A.2 Environment of tests

A.2.1 Population under test

Altogether, 20 persons from 72 to 97 years old, with a median age of 82 years old, participated to the test within four groups. The tests were realized in two different care centres for elderly people, during their activity time.

A.2.2 Equipments

- The main equipment was a MSI Top "all-in-one" touch-screen PC, on which the two interfaces had been deployed.
- For measurement concerns, the "CamStudio" software was installed on the PC to record the actions of the user on the GUI, and a camera filmed the activity of the person.
- A slideshow, made of three slides, was used to present the touch-screen pc, and how to use it.
- Several questionnaires were set up to collect different information, and guide the tests: personal information(age, sex, school level, etc.), skills and fears with relation to computers, seven scenarios to measure interfaces, and a last questionnaire about how they felt during the test, and how they perceived EnTiMid and its interfaces. These questionnaires, in French, are available in the report [CC09].

A.3 Protocol of test

1. First of all, people are asked to answer the first questionnaire about personal information, and the one about their skills and fears.
2. Once the first questionnaires is answered, the slideshow is played in interaction with the elderly person. Guided by the test driver, the person has to move forward the slides, by pressing a button on the touch-screen, in order for the person to familiarize with this technology.
3. The next step is a short training on the user interface under test. In a couple of minutes, the test driver presents the different controls, how to use them, what they can be used for, and where to find important information such as the current location.
4. The real test phase of the protocol is then engaged. The person is asked to realize a sequence of seven actions. These actions ask whether to act on the interface, or to get an information from the system.

5. The questionnaire about their feelings, and the use of these interfaces, is filled at the end of each test.

A.4 Threats to validity

The study had been led with only 20 elderly people. It is not sufficient to validate the GUI for a large-scale deployment, but it is for a preliminary work.
Moreover, the part of the population targeted by the IDA project is considered to be able to stay at home with a little help. These persons are thus living at home, and not in a care centre for elderly people. The conditions in which this study was led can thus be considered as too strict with relation to the real part of population targeted by the project.

A.5 Results and conclusion

The table A.1 presents the results of the tests. The time values displayed are all in seconds, and represent the average values of execution of each task. The total execution time column reports the average time necessary for people to complete the seven tasks of questionnaire. The other columns are the average time of execution, used to answer the particular questions of the questionnaire for each widget.

		Total Execution Time(s)	Heating Widget	Shutter Widget	Light Widget
Function/Zone	Phase 1	620,83	96	35,3	12,3
	Phase 2	238,75	14,25	16,75	6,25
Zone/Function	Phase 1	697	74	44,5	8
	Phase 2	429,6	37,4	36,8	8

Table A.1: Efficiency of use with relation to interfaces types

Significant improvement from Phase 1 to Phase 2
The total average execution time has been reduced of 62% in the Function/Zone mode, and 38% in the other mode thanks to the widgets improvements.

Function/Zone better than Zone/Function ?
The answer is YES. The Function/Zone navigation mode is more efficient than the Zone/Function mode. Whatever the phase of the project, people have always been faster using the Function/Zone mode, according to the data collected. This can be due to the radical change that occurs when passing from a function to another, which is not the case when changing of zone. Indeed, in each zone the widgets are the same, and only a label changes. All the widgets are changing from a function to another, making it easier to understand what is the current function, and then, identify where to activate

this function.

I would like to highlight the huge difference in execution time, between the Phase 1 - Zone/Function interface, and the Phase 2 - Function/Zone interface. It is a gain of 458,25 seconds, thus 7 minutes 38 seconds.
As a conclusion, the ITI project helped in validating three things. (1) It is more efficient to use a Function/Zone navigation than a Zone/Function. (2) The widgets are now ready. (3) The sketched application GUI can be used by elderly people on a touch-screen device, which answers the accessibility and usability requirement.

Acronyms

AAL Ambient Assisted Living. 18, 21, 27, 28, 104, 109, 110, 133, 135, 152
ADL Architecture Description Language. 51
AST Abstract Syntax Tree. 83, 84

DLNA Digital Living Network Alliance. 96
DPWS Device Profile for Web Services. 34, 96, 112, 120, 123
DSL Domain-Specific Language. 49, 51, 52, 58, 83, 145
DSPL Dynamic Software Product Line. 144

ESB Enterprise Service Bus. 42

GUI Graphical User Interface. 151, 154–156

HVAC Heating Ventilation Air-Conditioning. 20

ICT Information and Communication Technology. 27–29, 109, 110
IDA Innovation Domicile Autonomie. 109, 139, 141
IoS Internet Of Services. 37
IoT Internet Of Things. 38

JBI Java Business Integration. 42

MDE Model Driven Engineering. 49, 52, 57, 84, 136
MOM Message-Oriented Middleware. 71

PaaS Platform as a Service. 38
PLC Power Line Communication. 20, 31–33, 113

RFID Radio Frequency IDentification. 114
RPC Remote Procedure Call. 46

SaaS Software as a Service. 38
SCA Service Component Architecture. 22, 46, 47

SIP Session Initiation Protocol. 35

SOA Service Oriented Application / Architecture. 46, 48, 57, 59

SPL Software Product Line. 143, 144

UDDI Universal Description Discovery and Integration. 39, 40

UPnP Universal Plug & Play. 34, 96, 112, 120, 123

WSDL WebService Description Language. 40

Bibliography

[ADN+10] Françoise André, Erwan Daubert, Grégory Nain, Brice Morin, and Olivier Barais. F4Plan: An Approach to build Efficient Adaptation Plans. In 7th International ICST Conference on Mobile and Ubiquitous Systems (MobiQuitous'10), Sydney, Australia, December 2010. short paper.

[All05] HomePlug Powerline Alliance. Homeplug av white paper. White paper, HomePlug Alliance, 2005.

[All11] The OSGi Alliance. Osgi service platform core specification, release 4.3, 2011. `http://www.osgi.org/Specifications/HomePage` Sept. 2011.

[Arn] Terrell E. Arnold. Terrorism and the fear market. `http://www.rense.com/general49/fear.htm` Sept. 2011.

[ASS10] ASSAD. Les travaux ida - phase 1. Technical report, IDA, Juin 2010. `http://www.ida-autonomie.fr/file/ida-rapport_juin%202010.pdf` Sept. 2011.

[BBB+11] Arnaud Blouin, Morin Brice, Olivier Beaudoux, Grégory Nain, Patrick Albers, and Jean-Marc Jézéquel. Combining Aspect-Oriented Modeling with Property-Based Reasoning to Improve User Interface Adaptation. In ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pages 85–94, Pise, Italie, June 2011.

[BCJ+10] Benjamin Bertran, Charles Consel, Wilfried Jouve, Hongyu Guan, and Patrice Kadionik. SIP as a Universal Communication Bus: A Methodology and an Experimental Study. In International Conference on Communications (ICC'10), pages 1 –5, Cape Town South Africa, 05 2010.

[BCL+06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. Softw. Pract. Exper., 36:1257–1284, September 2006.

[BCU+04] Gordon Blair, Geoff Coulson, Jo Ueyama, Kevin Lee, and Ackbar Joolia. Opencom v2: A component model for building systems software. In IASTED Software Engineering and Applications, USA, 2004.

[BDLM11] Johann Bourcier, Ada Diaconescu, Philippe Lalanda, and Julie A. McCann. Autohome: An autonomic management framework for pervasive

home applications. ACM Trans. Auton. Adapt. Syst., 6:8:1–8:10, February 2011.

[Bos98] Xavier Bosch. barcelona investigating the reasons for spain's falling birth rate. The Lancet, 352(9131):887, 9 1998.

[BPKP10] Pyrros Bratskas, Nearchos Paspallis, Konstantinos Kakousis, and George A. Papadopoulos. Applying utility functions to adaptation planning for home automation applications. In Information Systems Development, pages 529–537. Springer US, 2010.

[Bra04] Gilad Bracha. Pluggable type system. In OOPSLA04 Workshop on Revival of Dynamic Languages, 2004.

[BRLM09] Yérom-David Bromberg, Laurent Réveillère, Julia Lawall, and Gilles Muller. Automatic generation of network protocol gateways. In Jean Bacon and Brian Cooper, editors, Middleware 2009, volume 5896 of Lecture Notes in Computer Science, pages 21–41. Springer Berlin / Heidelberg, 2009.

[BS93] J.M. Christian Bastien and Dominique L. Scapin. Ergonomic criteria for the evaluation of human-computer interfaces. Technical Report RT-0156, INRIA, June 1993.

[CBBJ08] Franck Chauvel, Olivier Barais, Isabelle Borne, and Jean-Marc Jezequel. Composition of qualitative adaptation policies. In Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE '08, pages 455–458, Washington, DC, USA, 2008. IEEE Computer Society.

[CBC10] Damien Cassou, Julien Bruneau, and Charles Consel. A Tool Suite to Prototype Pervasive Computing Applications (Demo). In Proceedings of the 8th IEEE Conference on Pervasive Computing and Communications (PERCOM'10), pages 1–3, Mannheim Germany, 2010. IEEE Computer Society Press.

[CBCL11] Damien Cassou, Emilie Balland, Charles Consel, and Julia Lawall. Leveraging Software Architectures to Guide and Verify the Development of Sense/Compute/Control Applications. In Proceedings of the 33rd International Conference on Software Engineering (ICSE'11), pages 431–440, Honolulu, United States, 2011. ACM.

[CBJ10] Mickael Clavreul, Olivier Barais, and Jean-Marc Jézéquel. Integrating Legacy Systems with MDE. In ICSE'10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops, volume 2, pages 69–78, Cape Town, Afrique Du Sud, 2010.

[CC09] Elsa COLAS and Nicolas COURTAIS. Intervention pour optimiser l'ergonomie d'une interface tactile intuitive facilitant le maintien à domicile des personnes handicapées et les personnes âgées dépendantes. Technical report, Université Rennes 2, LOUSTIC, Juillet 2009.

[CCQS05] Paolo Ciccarese, Ezio Caffi, Silvana Quaglini, and Mario Stefanelli. Architectures and tools for innovative health information systems: The guide project. International Journal of Medical Informatics, 74(7-8):553 – 562, 2005. MedInfo 2004.

[CCSV07] Ivica Crnkovic, Michel Chaudron, Séverine Sentilles, and Aneta Vulgarakis. A classification framework for component models. In Proceedings of the 7th Conference on Software Engineering and Practice in Sweden, October 2007.

[CDT08] Humberto Cervantes, Didier Donsez, and Lionel Touseau. An architecture description language for dynamic sensor-based applications. In Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE, 2008.

[CGFP09] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer, 42:37–43, 2009.

[CH04] Humberto Cervantes and Richard S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. In Proceedings of the 26th International Conference on Software Engineering, ICSE '04, pages 614–623, Washington, DC, USA, 2004. IEEE Computer Society.

[Cha04] David A. Chappell. Enterprise service bus, 2004.

[Com] European Commission. Fp7 tomorrow's answers start today. `http://ec.europa.eu/research/fp7/pdf/fp7-factsheets_en.pdf` Sept. 2011.

[CSMP07] Carlos Cetina, Estefania Serral, Javier Munoz, and Vicente Pelechano. Tool support for model driven development of pervasive systems. In Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES '07, pages 33–44, Washington, DC, USA, 2007. IEEE Computer Society.

[DDNDM07] Davide Devescovi, Elisabetta Di Nitto, Daniel Dubois, and Raffaela Mirandola. Self-organization algorithms for autonomic systems in the selflet approach. In Proceedings of the 1st international conference on Autonomic computing and communication systems, Autonomics '07, pages 26:1–26:10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[DFD+09] John Domingue, Dieter Fensel, John Davies, Rafael González-Cabero, and Carlos Pedrinaci. Towards the Future Internet - A European Research Perspective, chapter The Service Web: a Web of Billions of Services, pages 203–216. IoS Press, 2009.

[DMC09] Zoé Drey, Julien Mercadal, and Charles Consel. A Taxonomy-Driven Approach to Visually Prototyping Pervasive Computing Applications. In 1st

IFIP Working Conference on Domain-Specific Languages, volume 5658, pages 78–99, Oxford United Kingdom, 2009.

[DNGM+08] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering, 15:313–341, 2008.

[EHL07] Clement Escoffier, Richard S. Hall, and Philippe Lalanda. iPOJO: an extensible service-oriented component framework. Services Computing, IEEE International Conference on, 0:474–481, 2007.

[Fai06] Anthony Faiola. When escape seems just a mouse-click away - stress-driven addiction to online games spikes in s. korea. Washington Post Foreign Service, May 2006.

[FHL+11] Nicolas Ferry, Vincent Hourdin, Stéphane Lavirotte, Gaëtan Rey, Michel Riveill, and Jean-Yves Tigli. Ubiquitous Computing, chapter 8 - WComp, a Middleware for Ubiquitous Computing, pages 151–176. InTech, February 2011.

[Fie00] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. PhD thesis, University Of California, Irvine, 2000. AAI9980887.

[GMK02] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In Proceedings of the first workshop on Self-healing systems, WOSS '02, pages 33–38, New York, NY, USA, 2002. ACM.

[INPJ09] Paul Istoan, Grégory Nain, Gilles Perrouin, and Jean-Marc Jézéquel. Dynamic software product lines for service-based systems. In 9th IEEE International Conference on Computer and Information Technology, Xiamen, CHINA, October 2009.

[JMS05] François Jammes, Antoine Mensch, and Harm Smit. Service-oriented device communications using the devices profile for web services. In MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, pages 1–8, New York, NY, USA, 2005. ACM.

[joi] Ambient assisted living joint programme. `http://www.aal-europe.eu/about-us` Nov. 2011.

[JRS+09] Manuel Jimenez, Francisca Rosique, Pedro Sanchez, Barbara Alvarez, and Andres Iborra. Habitation: A domain-specific language for home automation. IEEE Software, 26:30–38, 2009.

[KAB+11] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. ACM Comput. Surv., 43:21:1–21:44, April 2011.

[LLC07] Marc Léger, Thomas Ledoux, and Thierry Coupaye. Reliable dynamic reconfigurations in the fractal component model. In ARM '07: Proc of the 6th international workshop on Adaptive and reflective middleware, pages 1–6, Newport Beach, CA, 2007. ACM.

[MAO+09] Nagy Michal, Katasonov Artem, Khriyenko Oleksiy, Nikitin Sergiy, Szydlowski Michal, and Terziyan Vagan. Automation Control - Theory and Practice, chapter Challenges of Middleware for the Internet of Things. InTech, December 2009.

[MB05] Selma Matougui and Antoine Beugnard. How to implement software connectors? a reusable, abstract and adaptable connector. In Lea Kutvonen and Nancy Alonistioti, editors, Distributed Applications and Interoperable Systems, volume 3543 of Lecture Notes in Computer Science, pages 1065–1069. Springer Berlin / Heidelberg, 2005.

[MBNJ09] Brice Morin, Olivier Barais, Gregory Nain, and Jean-Marc Jezequel. Taming dynamically adaptive systems using models and aspects. In Proceedings of the 31st International Conference on Software Engineering, ICSE '09, pages 122–132, Washington, DC, USA, 2009. IEEE Computer Society.

[Men01] Jerry M Mendel. Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions, volume 2. Prentice-Hall, 2001.

[MFB+08] Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jézéquel, Arnor Solberg, Vegard Dehlen, and Gordon Blair. An aspect-oriented and model-driven approach for managing dynamic variability. In In Proceedings of ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 08), Toulouse, France, October 2008.

[MFJ05] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In S. Kent L. Briand, editor, Proceedings of MODELS/UML'2005, volume 3713 of LNCS, pages 264–278, Montego Bay, Jamaica, October 2005. Springer.

[MNBJ09] Brice Morin, Grégory Nain, Olivier Barais, and Jean-Marc Jézéquel. Leveraging Models From Design-time to Runtime. A Live Demo. In 4th International Workshop on Models@Run.Time (at MODELS'09), Denver, Colorado, USA, Oct 2009.

[Mor10] Brice Morin. Leveraging Models from Design-time to Runtime to Support Dynamic Variability. PhD thesis, Université Rennes 1, Septembre 2010.

[MP06] Javier Muñoz and Vicente Pelechano. Applying software factories to pervasive systems: A platform specific framework. In 8th International Conference on Enterprise Information Systems (ICEIS 2006), May 2006.

[MPC06] Javier Muñoz, Vicente Pelechano, and Carlos Cetina. Implementing a pervasive meetings room: A model driven approach. In International

Workshop on Ubiquitous Computing (IWUC 2006), Cyprus, pages 13–20, May 2006.

[MRRS10] Rémi Mélisson, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. Supporting Pervasive and Social Communications with FraSCAti. In 3rd Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (DisCoTec'10), Amsterdam, Pays-Bas, June 2010.

[MRRS11] Rémi Mélisson, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. An SCA-based approach for Social and Pervasive Communications in Home Environments. Scientific Annals of Computer Science, XXI:151–173, 2011.

[MSCP06] Javier Muñoz, Estefania Serral, Carlos Cetina, and Vicente Pelechano. Applying a model-driven method to the development of a pervasive meeting room. ERCIM News, (65):44–45, April 2006.

[MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, 26(1):70–93, January 2000.

[NBFJ09] Grégory Nain, Olivier Barais, Régis Fleurquin, and Jean-Marc Jézéquel. Entimid : un middleware aux services de la maison. In 3ème Conférence Francophone sur les Architectures Logicielles (CAL'O9), Nancy, France, March 2009.

[NDBJ08] Grégory Nain, Erwan Daubert, Olivier Barais, and Jean-Marc Jézéquel. Using mde to build a schizophrenic middleware for home/building automation. In Proceedings of the 1st European Conference on Towards a Service-Based Internet, ServiceWave '08, pages 49–61, Berlin, Heidelberg, 2008. Springer-Verlag.

[NFM+10] Grégory Nain, François Fouquet, Brice Morin, Olivier Barais, and Jean-Marc Jézéquel. Integrating IoT and IoS with a component-based approach. In Procedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010), Lille, France, 2010.

[NT07] Jin Nakazawa and Hideyuki Tokuda. A middleware framework for sharing sensor nodes among smart spaces. In Fourth International Conference on Networked Sensing Systems (INSS '07), pages 171 –178, june 2007.

[OP97] A. Jefferson Offutt and Jie Pan. Automatically detecting equivalent mutants and infeasible paths. Software Testing, Verification and Reliability, 7(3):165–192, 1997.

[otEU] Official Journal of the European Union. Decision n. 743/2008/ec of the european parliament and of the council. `ftp://ftp.cordis.europa.eu/pub/fp7/art169/docs/aal.pdf` Sept. 2011.

[PACJ+08] Matthew M. Papi, Mahmood Ali, Telmo Luis Correa Jr., Jeff H. Perkins, and Michael D. Ernst. Practical pluggable types for Java. In Proceedings

of the 2008 International Symposium on Software Testing and Analysis (ISSTA'08), pages 201–212, Seattle, WA, USA, July 22–24, 2008.

[Pap03] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In Proceedings of the Fourth International Conference on Web Information Systems Engineering, Roma, Italy, WISE '03, pages 3–13, Washington, DC, USA, December 2003. IEEE Computer Society.

[RCAM+05] Anand Ranganathan, Shiva Chetan, Jalal Al-Muhtadi, Roy H. Campbell, and M. Dennis Mickunas. Olympus: A high-level programming model for pervasive computing environments. IEEE International Conference on Pervasive Computing and Communications (PerCom'05), 0:7–16, 2005.

[RHC+02] Manuel Román, Christopher Hess, Renato Cerqueira, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. IEEE Pervasive Computing, 1:74–83, 2002.

[RHT+10] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. Restful integration of heterogeneous devices in pervasive environments. In Distributed Applications and Interoperable Systems (DAIS'10), volume 6115 of Lecture Notes in Computer Science, pages 1–14. Springer Berlin / Heidelberg, 2010.

[RM09] Romain Rouvoy and Philippe Merle. Leveraging component-based software engineering with fraclet. Annals of Telecommunications, 64:65–79, 2009.

[RvdLKM00] RobVanOmmering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics. IEEE Software Computer, 33(3), pages 78–85, March 2000.

[sca] Sca specifications. `http://www.osoa.org`.

[SML+10] Andrew Sixsmith, Sonja Mueller, Felicitas Lull, Michael Klein, Ilse Bierhoff, Sarah Delaney, Paula Byrne, Sandra Sproll, Robert Savage, and Elena Avatangelou. Intelligent Technologies for Bridging the Grey Digital Divide, chapter A User-Driven Approach to Developing AAL Systems for Older People:The SOPRANO Project, pages 30–45. IGI Global, 2010.

[SVP10] Estefanía Serral, Pedro Valderas, and Vicente Pelechano. Supporting runtime system evolution to adapt to user behaviour. In Proceedings of the 22nd international conference on Advanced information systems engineering (CAiSE'10), pages 378–392, Berlin, Heidelberg, 2010. Springer-Verlag.

[TLR+09] Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Vincent Hourdin, Daniel Cheung-Foo-Wo, Eric Callegari, and Michel Riveill. Wcomp middleware for ubiquitous computing: Aspects and composite event-based web services. Annals of Telecommunications, 64:197–214, 2009.

[Tra09] Laurence Tratt. Dynamically typed languages. Advances in Computers, 77:149–184, 2009.

[Tra10] Laurence Tratt. A modest attempt to help prevent unnecessary static / dynamic typing debates. Technical report, Bournemouth University, UK, 2010.

[Tri08] Tridium. Niagaraax. Communications Magazine, IEEE, 46(12):22, december 2008.

[upn] The UPnP Forum. http://www.upnp.org.

[VAMC08] Dimitrios Vergados, Alevizos Alevizos, Anargiros Mariolis, and Michael Caragiozidis. Intelligent services for assisting independent living of elderly people at home. In Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments, PETRA '08, pages 79:1–79:4, New York, NY, USA, 2008. ACM.

[WSO+10] Peter Wolf, Andreas Schmidt, Javier Parada Otte, Michael Klein, Sebastian Rollwage, Birgitta König-Ries, Torsten Dettborn, and Aygul Gabdulkhakova. openAAL - the open source middleware for ambient-assisted living (AAL). In AALIANCE conference, Malaga, Spain, March 11-12, 2010.

[WSV+07] Marion Wiethoff, Sascha Sommer, Sari Valjakka, Karel Van Isacker, Dionisis Kehagias, and Evangelos Bekiaris. Specification of information needs for the development of a mobile communication platform to support mobility of people with functional limitations. In Constantine Stephanidis, editor, Universal Access in Human-Computer Interaction. Ambient Interaction, volume 4555 of Lecture Notes in Computer Science, pages 595–604. Springer Berlin / Heidelberg, 2007.

[YWDJ98] Joseph W. Yoder, Quince D. Wilson, Mcdonnell Douglas, and Ralph E. Johnson. Connecting business objects to relational databases. In Pattern Languages of Programs(PLOP'98), Monticello, Illinois, USA, volume 5, pages 51–89, August 1998.

[ZBB+07] Elmar Zeeb, Andreas Bobek, Hendrik Bohn, Steffen Prueter, Andre Pohl, Heiko Krumm, Ingo Lück, Frank Golatowski, and Dirk Timmermann. Ws4d: Soa-toolkits making embedded systems ready for web services. In 3rd International Conference on Open Source Systems, Embedded Workshop on Open Source Software and Product Lines, Limerick, Ireland, 2007.

List of Figures

1.1 S-Cube Research Framework 10

2.1 Median Age of EU Population - Source Eurostat 17
2.2 Age Pyramid EU (27) in 2009. - Blue: M, Green: F - Source Eurostat 18

3.1 Data from [DFD+09] and actualized from ProgrammableWeb.com . . . 38
3.2 WebService Architecture 40

5.1 Overview of the EnTiMid layers 66

6.1 Functional Interfaces 71
6.2 Configuration Phase 72
6.3 Example of Interoperability 73
6.4 Electronic Parallel: Datasheets 76
6.5 Extraction of a part of the component model architecture 77
6.6 Electronic Parallel: Components 78
6.7 Example Model 80
6.8 Link between the interoperability layer and component connections . . 85
6.9 Electronic Parallel: Simulation 87
6.10 Checkpoint positions in the assembly deployment chain 88
6.11 Identifying differences between the source and the target configurations. 92
6.12 Instance creation tool chain 95

7.1 EnTiMid development chain 100

8.1 Solution elements for Mrs P. 111
8.2 Equipments available for the study 112
8.3 Components used in the interoperability experiment 115
8.4 Components used in the evolution experiment 117
8.5 Components used in the adaptation experiment 119
8.6 Time (in ms) spent in Configuration Comparison and Actual Reconfiguration 119
8.7 Mapping UPnP-EnTiMid 122

9.1 S-Cube Research Framework 139

10.1 Science festival presentation 146

A.1 Results of the first phase 153
A.2 Improved interfaces, result of the second phase 153

VU :
Le Directeur de Thèse
(Nom et Prénom)

VU :
Le Responsable de l'Ecole Doctorale
(Nom et Prénom)

VU pour autorisation de soutenance

Rennes, le

Le président de l'Université de Rennes 1

Guy CATHELINEAU

VU après soutenance pour autorisation de publication :
Le Président de Jury,
(Nom et Prénom)

Résumé

Les systèmes logiciels tendent à se doter de facultés d'adaptation, d'évolution et d'ouverture. Ces capacités requièrent une grande flexibilité et dynamicité de l'environnement d'exécution, ainsi que de nouveaux outils d'assistance à la fabrication de ces systèmes. En électronique, des outils ont été déployés pour faire face à l'hétérogénéité et au nombre de composants, ainsi qu'aux besoins d'adaptation de produits existants à de nouvelles technologies. L'ouverture de la documentation et des spécifications a permis une grande richesse de solutions venant tant de bricoleurs que d'industriels. Inspiré par l'électronique, cette thèse contribue à l'amélioration de la flexibilité des systèmes logiciels tout en conservant un haut niveau de fiabilité. Les apports se font à trois niveaux.
(1) Un nouveau modèle de composants qui offre une grande flexibilité et permet la connection de composants hétérogènes.
(2) Des outils issus de l'ingénierie des modèles, pour créer, modifier, simuler et valider la structure et le comportement des assemblages de composants avant leur déploiement.
(3) Un environnement d'exécution bati sur une architecture à base de services, pour supporter les évolutions, les adaptations et l'ouverture requises par le modèle de composant proposé.
Cette thèse a été validée sur un cas concret dans un projet d'aide à domicile. Dans ce domaine, les systèmes logiciels doivent être adaptables et flexibles, pour répondre aux évolutions des besoins et pathologies des personnes âgées. Les bénéfices acquis de l'utilisation de cette approche dans ce contexte ont prouvé la pertinence de cette thèse.

Abstract

Software systems tend to acquire capabilities of adaptation, evolution and openness. These abilities require the execution environment to be highly flexible and dynamic, and require new tools to handle these abilities. In electronics, tools have been set up to cope with the huge heterogeneity and number of components, and the adaptation of existing products to new technologies. Openness of documentations and specifications in this area led to a wealth of solutions made by industrials or individuals fond of electronics. Inspired by electronics' achievements this thesis contributes in improving the flexibility of software systems while maintaining a high level of reliability. The contribution is threefold. (1) A new component model, which improves flexibility to enable connection of heterogeneous components. (2) Tools from model driven engineering, to create, edit, simulate and validate the structure and behavior of component assemblies prior to their (re-)deployment. (3) A runtime environment built on top of a service-based architecture to support evolutions, adaptations and openness required by the proposed component model.
This thesis has been validated on a use case from an Ambient Assisted Living project. In this domain, software systems have to be adaptive and flexible, to fit the needs and pathology evolutions of elderly people. Although there still is a long way to go, the benefits gained from the use of this approach in this context proved the relevance of this thesis.