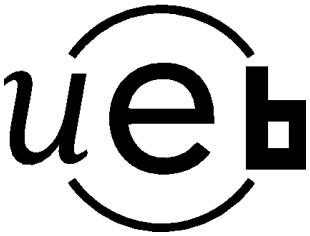
N° d'ordre : 4406 ANNÉE 2011



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 24 Octobre 2011

devant le jury composé de :

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

Charles CONSEL
Professeur, Université de Bordeaux 1 / *Rapporteur*

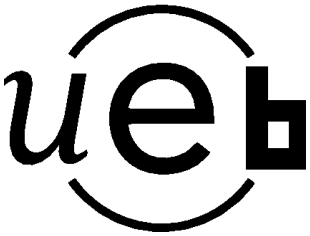
Elisabetta DI NITTO
Professor, Politecnico di Milano / *Rapporteuse*

Didier DONSEZ
Professeur, Université Joseph Fourier - Grenoble 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 2011



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

Thèse soutenue à Rennes
le 24 Octobre 2011

devant le jury composé de :

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

Charles CONSEL
Professeur, Université de Bordeaux 1 / *Rapporteur*

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

Didier DONSEZ
Professeur, Université Joseph Fourier - Grenoble 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

Coming soon !

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

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 à proprement parlée est décrite et discutée en termes d'adéquation aux besoins. Enfin, une mise en évidence des bénéfices immédiats et de quelques limitations identifiés 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.

L'ouverture et le contrôle à distance doivent 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 disponible 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.
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.

La gestion de la variabilité et la distribution sont des préoccupations liées au fort besoin de personnalisation des solutions, et à la dispersion des déploiements. En effet, l'ensemble des options déployées sur les systèmes à l'échelle d'une ville, peut très vite devenir ingérable. 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 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 et la sécurité sont des éléments important dans les systèmes domotiques. Ces systèmes se voient confiée 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 bloqués en cas d'urgence par exemple. Par ailleurs, les accès au système doivent être sécurisés afin d'éviter qu'il soit contrôlé par des personnes non autorisées, mais pas au point d'en faire une contrainte 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.

Les approches s'intéressant à la résolution de problèmes d'interopérabilité, d'adaptation ou de contrôle distant pour différentes applications, sont assez abondantes dans la littérature scientifique.
D'une manière générale, les approches s'appuyant sur des architectures à base de services semblent être adaptés 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, offrent pour leur part un niveau d'abstraction idéal pour représenter les produits domotiques. Cependant, les ports de communication des composants sont souvent définis par l'intermédiaires d'interfaces de programmation qui rendent impossible certaines connections non prévues à l'avance, sans ajout de connecteur *ad-hoc*.
Il semble toutefois que le mélange de l'approche à composant, et l'approche des architectutres à base de services, identifié comme des composants pour les architectures orientées services, soit l'approche qui réponde le mieux aux besoins identifiés ; que les bénéfices de l'une permettent de combler les manques de l'autre approche.
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 et faible 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 services, 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'affèrent chacun à résoudre une question particulière dans le problème global.

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

La couche **Modèle de composants** apporte les structures et méthodes nécessaires à la représentation et à la manipulation des produits, à un niveau d'abstraction adéquate. 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). Ces descriptions précises des composants, apportées par le nouveau modèle, permettent à la couche de modèle à l'exécution(*Model@Runtime*) de travailler dans les meilleurs conditions. Enfin, des outils ont été développés afin de garantir, en permanence, la cohérence entre les implémentations et leurs représentants dans le modèle.

A leur niveau, **le Model@Runtime et les Validateurs** s'occupent d'offrir des outils facilitant la gestion des composants et des assemblages à l'exécution. A ce niveau, les spécificités d'implémentation des composant sont effacées par le modèle de composant sous-jacent. 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.

Les **Wrappers** prennent en charge la publication des produits présents dans le système, sur des protocoles de niveaux applicatif. 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 de haut niveau.

L'**Environnement d'exécution orienté services** complète la contribution en apportant 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 détail les forces et faiblesses 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 et faiblesses comparées aux mêmes préoccupations.

		Interopérabilité	Ouverture	Adaptation	Évolution	Gestion de la Variabilité	Contrôle Distant
MDE /DSL	Habitation	+				+	
	DiaSuite	+	+				+
	PervML	+	+	+	+	+	
Modèles de composants	Darwin			+	+		
	Koala				+	+	
	Fractal			+			+
	uMiddle						+
Frameworks orientés services	Hydra	+	+	+			+
	JBI	+	+		+		
	OSGi			+	+		
	SOPRANO			+	+		+
Modèles de composants pour les services	SCA		+		+		+
	FraSCAti	+	+	+	+		+
	iPOJO			+	+		
	Gaïa	+		+	+		+
	Niagara	+	+				
	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, quand à elle, est facilitée par la pré-

sence du modèle pendant l'exécution, mais les outils restent insuffisants pour résoudre complètement le problème.

1.6 Conservation des bonnes propriétés

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 ynchronisé 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 aux raisonneurs de modifier les connexions, ou même les composants, pendant l'exécution. Le renforcement de l'indépendance des composants pour permettre l'interopérabilité et les adaptations, renforce aussi la nécessité d'extraire les dépendances.
Le **déploiement à chaud** fût conservé grâce au choix fait de baser les développements de l'environnement d'exécution sur une plateforme orientée services. Il était nécessaire de conserver cette propriété afin que les raisonneurs soient capables d'adapter le système, ou le faire évoluer, en déployant de nouveaux composants sans redémarrage.
L'isolation close entre les types de composants et les instances était 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 rendent le développement des composants plus simple. 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'utilisation d'annotations dans le code des composants, et la présence d'un générateur de code facilitent le travail des développeurs. 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.

1.7.2 Création d'applications simplifiée

Le modèle de composant et les outils de modélisation rendent simple 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 par des interactions utilisateurs comme le glisser-déposer. Les connections port-à-port permettent de connecter deux ports, quels que soient leurs noms ou utilités, si les outils de vérifications personnalisables autorisent la connections. Par ailleurs, les développeurs de produits domotiques, familiers des composants électroniques, sont vite familiarisés avec ce modèle de composants de par sa proximité du modèle électronique. Cette rapide prise en main permet aux ingénieurs et techniciens de créer facilement des applications personnalisées pour répondre strictement aux besoins de chaque personne.

1.7.3 Viabilité et précision

La possibilité de faire évoluer la solution, ou de l'adapter au cours du temps, renforcent sa pérennité des solutions une fois déployées. Ces facultés offrent au système la capacité de suivre les évolutions de la pathologie ou des technologies sans avoir à refaire 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 facile 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 ainsi se faire 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, pas comportementale

Le modèle de composant proposé dans la contribution de cette thèse facilite la description structurelle d'un système, alors que les gens sont plus en clin 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.
Mais le problème n'est pas simple, parce que le comportement global du système peut être décrit en plusieurs morceaux. En effet, les gens vont être 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 instables.
Enfin des utilisateurs non experts du domaine, et les utilisateurs finaux ne le sont pas à 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 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-goc*. Le modèle de composant proposé dans cette thèse ne s'est pas vraiment débarrassé de ce problème. Il a en fait 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, pouvant avoir des comportements plus complexes qu'un simple appel de méthode ou envoie de message. Des mécanismes de mapping ou de renommage, par exemple, peuvent être imaginés pour résoudre l'alignement.

1.8.3 Des vérifications légères

L'expérimentation n'a pas nécessité la mise en oeuvre de vérifications complètes ou complexes des modèles. Seulement quelques vérifications structurelles sur le modèle ont été implémentées pour rechercher des cycles par exemple. Beaucoup des points de vérifications n'ont pas été complétés, parce qu'ils sont dépendant 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. Aussi, il est probable que le modèle ne contienne actuellement pas toutes les informations nécessaires pour chacune des étapes de vérification.

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 facilement. 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.

1.8.5 Pas de test sur plateforme embarquée

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 beaucoup de plateformes embarquées. Dans un vrai déploiement, 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 plus embarquée suffise. Nous n'avons pas menés d'expérimentation en ce sens, mais l'implémentation étant faite en Java, la plateforme doit pouvoir accepter une machine virtuelle Java sans quoi, EnTiMid ne peut fonctionner.

1.9 Contribution au réseau d'excellence Européen S-Cube

La contribution de cette thèse d'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 hybrides 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 [1].
La contribution de cette thèse contient 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 al conception au déploiement pour assurer la qualité de service.

Plus précisément, la contribution de cette thèse d'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 d'S-Cube, EnTiMid pourrait être considéré pour participer à l'adaptation des couches d'infrastructure ou de composition et coordination de services, couches présentés sur la figure 1.3.2.

1. DoW Amendment 4, December 6th, 2010

Part I

Context, Requirements and State of Art

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 in AAL projects, Home Automation, and software engineering approaches is realized 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 the first section. This presentation enables the section 2.3 to list some general requirements identified in these domains. Lastly, section 2.4 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 regularly been growing. From a median age of 37.7 years in 1999, it raises 40.6 years in 2009 as shown on figure 1.2.1. 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.

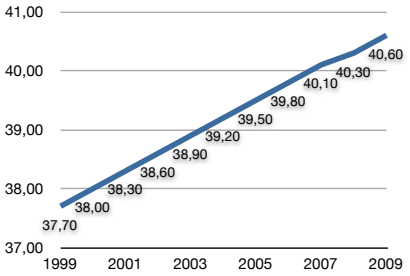


Figure 1.2.1: Median Age of EU Population - Source Eurostat

The **"Baby Boom"**
During the Second World War the birth rate stagnated, resulting in a similar number of birth from 1939 to 1945. This stagnation is visible on figure 1.2.2, at the level of people aged between 64 and 70. The "Baby Boom" describes the rapid and strong augmentation of the number of births that occurred after the Second World War, thus between

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

1945 and 1968. Actually, 4.9 million people were born in 1944 in EU, 7.6 million born in 1968 (plus 35.8%). People born during the "Baby Boom" are now(in 2011) 43 to 68 years old, and will soon retire.

Decrease of birth rates
As it can be noticed on figure 1.2.2, a decrease in birth rates started at the end of the sixties. From 7,664 million of persons born in 1968, the number of births fell down to 5,061 million in 2002 (minus 33.4%). In [Bos98], Xavier Bosch explains that this phenomenon, in Spain, 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 augmentation of the women part in the workforce.

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

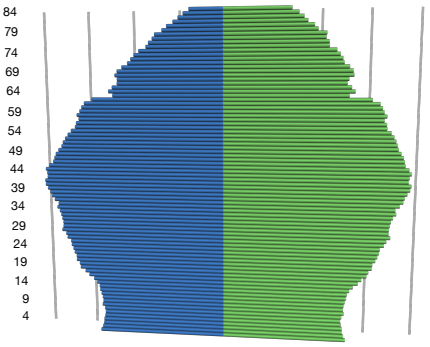


Figure 1.2.2: Age Pyramid EU(27) in 2009. - Blue:M, Green:F - Source Eurostat

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

2.1.2 The concept

The concept of Ambient Assisted Living (AAL) can be described through 6 dimensions.
Autonomy By increasing the autonomy, the self-confidence and the mobility of elderly people, AAL tends to extend the time people can live in their preferred environment.
Activities Maintaining physical or intellectual exercises helps elderly people to stay on a good health, and prevent from a decrease of capabilities.
Assist individuals at risk, by promoting a better and healthier lifestyle.
Secure Support and maintain the network around the individual, including family, friends and social activities, to enhance the security and prevent from social isolation.
Support carers, families and care organisations in their everyday activities.
Streamline the use of resources dedicated to elderly people, by increasing their efficiency and productivity.

There are many solutions to address these dimensions. Automation of some tasks can enforce the autonomy, and the use of mechanical helps can improve the mobility. Animators and health professionals can propose activities, support and assistance. Unfortunately, health care associations or companies have difficulties to hire people for these jobs. Indeed, qualified people are not enough numerous, and financial constraints are strong in this domain.
A workaround, in assisting both helped people and helpers, could be to use Home Automation technologies in conjunction with ICTs and human interventions. The section 2.2 presents this domain.

2.2 Home Automation

Do you remember the Jetsons ? This cartoon family was born in 1962, from William Hanna and Joseph Barbera. George and Jane were living in the Skypad Apartments in Orbit City with their children Judy and Elroy. Their housekeeping robot, Rose, was handling 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 well describes the idea of smart homes or home automation. Let us get further down into a bit more details about home automation, and see why it is still not applied in everyone's house.

Figure 1.2.3: The Jetsons

2.2.1 Application domains

Home automation has too long been perceived as presented in the Jetsons: a set of costly, useless, funny pieces of technology.
Obviously, personal home theatre, multi-room multimedia diffusion systems, smart coloured lighting controllers will always exist and make a good showcase of what is possible to do. Besides, a more utility-oriented home automation is soaring. Among others, home or building automation technologies 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 kind of systems in homes. The minimization of power consumption with a maximisation of welfare is acceptable, even dreamed by everybody. By extension, alarm systems, automatic garage doors, 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 through many aspects, but it is also a fantastic set of tools awaiting to be connected together.

Communication Media

Four main communication medium 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 address. For instance, a Bus link is highly reliable, but needs 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 are no home automation specific brand or manufacturer. They all come 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 used several communication media to carry the communications. A simple consequence of that is the huge number of devices, protocols and medium available on the home automation market. 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 on mono-manufacturer expensive close solutions, or no solution at all.

Even if home automation technologies have been existing for many years, no real standard has been released. Imposing a global standard, as it was the case for the IP protocol for example, seems to be the best way to address such an issue. Naturally, 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 rude task. And even if one finally emerges, developers will still have to deal with legacy devices using proprietary protocols.

Moreover, the old way manufacturers are thinking adds to the complexity of the problem. They still think that a close 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 to be 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 to 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 marginal the adoption and use of these technologies. This absence of 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 had been identified in [NBFJ09]. This section completes the list and precise 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 to ensure each of them to be operable by another one ? For instance, if I have a light management system and another one for shutters, how to guarantee these systems to be able to communicate in order to be of use for the user.
A close environment, in which all devices come from the same manufacturer, avoids the problem. A communication interface, common to all components, also answers the question. The 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 his diseases. Indeed, specialists in this domain will select different technical artefacts or services, according to each need of the person. Those elements of solution will have to cooperate one to each other in a single system, to perfectly fit the user's needs, and this, whoever their manufacturer or whatever their communication protocol may be.

Openness
By openness, here is meant to make all offered functionalities available for third party applications, or different uses than the one basically thought. Too many times, systems are closed, and providers impose their clients to ask them for any evolution, even if another provider could do it. Keeping the system open to external contributions may indeed pull down the number of demands for system modifications or add-ons. Anyway, it is also a door for new unforeseen appliances adding a smart behaviour on top of a reliable set of functionalities.

This is a clear challenge in computer sciences, and in general when considering communications and interactions between objects or systems. 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, computer manufacturer would not create a specific connection port, not compatible with all other computers. Unless they want to create a new captive market.

Adaptation
In an ideal world, devices never fail, services are always available, and Internet is always accessible. 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 consider their execution environments. 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. Obviously, these adaptations should not require any reboot of the system, because the 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.
Policies of execution or reactions of the system 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, automate the management of certain household functions such as heating or shutters management. This kind of change in the system behaviour 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 basic security functions execution (like emergency calls handling), neither in their behaviour, nor in their execution during adaptation.
Adaptations do not only concern technical elements, but also the user himself. 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 improve some memory loss. Where some have vision troubles, others experiment 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 comfort of the inhabitants, have to be designed to serve during all the life of the building (even if hardware changes may be required). Therefore, they must be ready to accept future and unforeseen evolutions like installation of new services/functionalities for ex-

ample.

Variability Management
No house or building is like any other. Because of structural or ground specificities, because of particular user's need or just because no one would like to have exactly the same house as his neighbour. 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 excluded. 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 solutions for an installation query, should have at their disposal some tools to help them in choosing devices to install, or assist them in devices selection from all the catalogues of all manufacturers.

Remote Control
More and more, people want to remotely access to their belongings. This is also true for their homes. They would like to be able to remotely check lights' states, run a bath or switch on the alarm system 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 a remote access to all systems, or check locally by hand. Obviously, the check by hand solution is no more 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 distant diagnosis, corrections, or evolution actions from a control centre.

Distribution
Today's systems are more and more working with or on different execution platforms, and communicate 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 ensure the interim, until the original controller is back. Also, devices are getting smarter and may make decisions by themselves. As a consequence, decisions and control could be distributed on several smart equipments.

Safety & Security
As presented in the context of this work, safety and security are very important requirements for home automation systems. Actually, they have to be themselves 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 stay stuck in the house in case of emergency. Moreover, access policies of the system have to be sufficient to avoid unauthorised access, and quiet enough to not become a constraint for authorized carers.

Several tools like simulators, tests, or model checking, can be used to check and improve the reliability and the safety of such kind of systems.

Acceptability & Accessibility
In the domain of home automation, deployed solutions, whatever their nature, have to consider interactions with all inhabitants. Being children, teenagers, young active, parents, retired or old people, all of them must be able to interact with the system. Checks have to be performed on any solution proposed, to ensure all users can control, or get information from the system, and keep the control.

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

2.4 Contribution of this thesis

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

In the electronic domain, the number of components, and their always-possible connectivity have offered technicians and engineers, means to create various solutions. Even many years after their assembly, electronic devices can still be repaired or completed with new features. The proposition 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 by three aspects :

- A constraint-relaxed component model that leverage the software flexibility, by offering ways to connect any component to any other. This aspect addresses issues from interoperability and evolution requirements
- Modelling tools to create, modify and simulate component assemblies, check their consistency and validity before their (re-)deployment at runtime. Safety and security, as long 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 requirements of adaptation and evolutions at runtime.

Chapter 3

State of Art

The section 3.1 starts with a description the Ambient Assisted Living domain from which social requirements have emerged, and 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 underlying needs targeted by this thesis.
Section 3.2 gets the survey underway with Model Driven Engineering (MDE) and Domain-Specific Language (DSL). Some component models are studied in section 3.3 and section 3.4 describes some service-based tools. The two last sections are addressing component models for Service Oriented Application / Architecture (SOA) and resource oriented architectures.

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 grouped in the AAL Association, which main objective is to enhance the quality of life of elderly people, by the use of Information and Communication Technology (ICT). Their main activity is to found R&D projects in the AAL domain, and to publish annual calls for project proposals.
The total budget planned for the AAL Joint Programme is 700M€, funded by public resources(AAL Partners plus European Commission) for a half, and by private participating organizations for the rest. According to the 13th article of 743/2008/EC [otEU] EC decision, funding from European commission are taken from the budget allocated

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

to the ICT theme of the 'Cooperation' Specific Programme of FP7. Thus, the AAL-JP is set set last from 2008 to 2013, to fit the FP7 dates.

The ASK-IT project [2] ran from 2004 to 2008, and aimed at providing *Ambient Intelligence* to support and promote mobility of impaired people [WSV+07]. They developed a software framework, and a set of tools for mobility. Among others, a *Domestic Module* has been created to support the provision of seamless home environment management, to the mobility-impaired traveller 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 led to measure the acceptability of the solutions. SOPRANO [SML+10] aims to maximise the acceptability of the services, especially in populations vulnerable to loss of independent living. This is achieved by ensuring the maintain of the control on the environment by users, where they wish to, and using extended user-centred design techniques.

In-Home IST Project [4] is also an interesting project in this context. Ended in December 2008, its goal was to enhance the autonomy and the safety for elderly people at home [VAMC08]. They developed a set of services like activity monitoring, home environment management, tasks scheduling, flexible AV streams handling, or hence, flexible access to household appliances.

On its side, the GUIDE project [5] tends 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 help. 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 by section 2.2.

3.1.2 European research

The European 'Cooperation' programme of research of the FP7 is quite a good representative of European countries' current preoccupations and research axes. According to the 7th Framework Programme factsheet[Com], 32.365M€ 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

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)

positions are respectively Transport and Nano-Productions with 13,18% and 11,03%, followed by Energy and Environment themes consuming 7,25% and 5,67% of the overall budget.
Thus, three of the top five preoccupations of Europe are ICT, Health, and Energy and Environment(assuming Energy and 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 watermark on several projects in 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 have been partly collected from a talk by Luc BALANGER, director of the Communication Networks department at FFIE (French Federation of Electricians).

Asian countries developed a strong market sensitivity to video games. Some TV channels are even specialized on live transmission of gaming parties, involving professional and sponsored gamers. Over the past few years, several studies and news reported a kind of addiction of a part of the population to video games. In particular, A. Faiola in [Fai06] states that about 2,4% of 9 to 39 years 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.
Home automation manufacturers are thus working on products making the game more real. The goal is to give gamers the sensation of being into the game, and play 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 are 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. Safety and security of people and goods has been a huge market in the United States since 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, security video camera 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, figuring there is somebody at 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, the nature is not clement with 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

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

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, remotely run a bath using a smart phone for it to be ready when 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

Evolutions of concerns and needs in the home automation domain 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 manufacturer, and to their domain of use, devices also come with their own transportation protocols making the diversity even greater. This section briefly presents a non exhaustive list of protocols and media used by home automation products to enlighten the complexity of this domain.

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. The probably most famous ambassador of this communication medium is Ethernet.

PLC is a special kind of bus. The main idea of this communication medium is to use again 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 on the power line by modulating in a carrier wave. All transceivers plugged on 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 conceive, because of the trade off between energy consumption and protocol reliability. However their deployment is easy, and does not require a heavy work. Many manufacturers are using 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), 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 like 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 are using several communication media. Obviously, they also created products to transfer orders/communication frames from one media to another, for all their products to get the information.

Transportation Protocols

IO-Homecontrol is a two-way radio technology, and a proprietary protocol. Organized in an association, a dozen of industrial manufacturers are providing 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 equipments of the house like roof window, vertical window, roller shutter, gate, garage door, front door, alarm system, lighting, ventilation, heating system, 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 aim to 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 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.

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 are offering interoperable products in these domains, using this close 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.

European Eib/KNX consortium The KNX Association [8] has been 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 recognised 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 [9] 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. [10] organization. This organization is also responsible for giving guidelines and help, to manufacturers, integrators and end-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 guideline 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). As for 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).

7. http://www.z-wavealliance.org/
8. http://www.knx.org
9. http://www.echelon.com
10. http://www.lonmark.org

BACnet Developed under the auspices of the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), BACnet [11] 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 are working to integrate issues from various building aspects from "Objects and Services" collaborations to elevator or lighting management. As for February 25, 2011, 503 Vendor IDs have been issued from all over the world.

X10 is a rather simple communication protocol for low cost home automation products. Sensors and actuators are sending 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 and design of compatible products is rather simple.

ZigBee is an open standard addressing low-cost, low-power M2M wireless networks. It has been released by the IEEE in 2003, and over 300 manufacturers have then join into 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 are targeting remote control, light management or sensing domains.

6lowPan is the name of a working group at the Internet Engineering Task Force(IETF). This group aims at reducing 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 embedded distributes applications, making them operable from any classical IP network. Specifications of 6LowPan can be found as RFC 4944 [12] and RFC 4919 [13].

Application Protocols

Universal Plug&Play(UPnP)
Universal Plug & Play (UPnP) [14] is an application level protocol which aims to ease the connection and use of electronic devices, based on a discovery-search mechanism. As an UPnP-Device joins the UPnP network, it sends an XML description file to all

11. http://www.bacnet.org
12. http://tools.ietf.org/html/rfc4944
13. http://tools.ietf.org/html/rfc4919
14. http://www.upnp.org

UPnP-ControlPoints. 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-Devices are also specified in its description. Each service conforms to a type, a set of UPnP-Actions the service renders. Providing such information to other devices on the network makes it possible to use functionalities of the device without having to install anything (thanks to the standardization of the descriptions and method call mechanisms).

Device Profile for Web Services (DPWS)
DPWS [JMS05] could be considered as a UPnP next generation. If the goals are the same, DPWS is using WebServices as a transportation mechanism where UPnP uses simple XML over IP. DPWS still based on the concept of service, device, operation and parameter. It includes numerous extension points, allowing for seamless integration of device-provided services in enterprise-wide application scenarios.

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 MMUSIC Working Group, it was initially published in 1996 as RFC 2543 and actualized 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 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 request and answer messages, and can use again many concepts of previous standards like HTTP and SMTP.

XMPP
The Extensible Messaging and Presence Protocol (XMPP) [15] 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. Last reviews of RFCs 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 produce a wide variety of XMPP-based software.

The complexity of the home automation domain clearly appears from this presentation. Taken apart, each product, transportation protocol, or medium is quite easy

15. http://xmpp.org/ (May 2011)

to handle. The complexity is due to the huge number of possible solutions for a given problem, and to the difficulty of getting aware of all benefits and drawbacks brought by each solution.
Home automation can be used in various contexts, like assisted living or energy 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 the variability management, and conciliate home automation technologies with domain specific problems to bring new solutions.

3.2 Model Driven Engineering & Domain Specific Languages

3.2.1 Description

MDE is an approach that promotes the use of an abstract representation of a 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 predefined requirements. Models are human understandable representations of the reality. They can handle information about the structure, the data exchange, the communication links, or some building constraints of a software.
DSL are another mean 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 their requirements, by using their own terminology.

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

3.2.2 Projects

3.2.2.1 Habitation

Habitation is a methodology, a set of tools to address specific requirements of home automation application development and design. In [JRS+09], Jimenez et al. describe how the combination of a DSL, and a MDE approach, eases the creation of solutions in this domain. Habitation proposes three main tools. A catalogue 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 catalogue can be placed to define a specific application. Called the application view, this tool aims to simplify the assembly work to 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 easy to handle. However, Habitation does only address pre-deployment design issues, and does not treat issues such as evolutions, hot deployments, remote control means, in short, runtime considerations.

In all this section, each approach or tool comes with a table. This table presents, in a synthetic way, strengths and weaknesses of each tool. Individual tables are merged altogether in section 4 as a synthesis.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	-	-	-	+	-

3.2.2.2 DiaSuite

DiaSuite [16] 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 realized with DiaSuite tools.
This tool-suite has been augmented with Pantagruel [17], a visual DSL created to simplify again, 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 generated an application code, compatible with the DiaSuite tools. Details about this tool are available in [DMC09].

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

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	+				+

16. http://phoenix.inria.fr/projects/diasuite
17. https://pantagruel.bordeaux.inria.fr/

3.2.2.3 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 3.4.2.2) 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 great 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 made simpler 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 Remote control are not treated in this approach.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+		+	+		

3.2.2.4 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 view, describing the requirements of the system at a high level of abstraction, and the architect 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 3.4.2.2) code. This tool chain, detailed in [MP06], firstly transforms the platform independent PervML model to a OSGi dependent model, 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 evolutions capabilities to adapt the generated systems to changes in the 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 naturally offers adaptation, evolution, openness and interoperability mechanisms. As presented in [CGFP09], PervML also target 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. Remote control is not addressed in this work.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	+	+	+	+	-

MDE and DSL are handy tools to ease the description and high level representation of software systems. Often based on generative approach, they can cover a large part of the software development chain. However, it is sometimes convenient to reuse existing software parts for several reasons. Reliability of a several time experienced piece of code, memory space reduction, or time saving argue in this sense. The promotion of reusable software pieces is notably pulled by software components.

3.3 Component models

3.3.1 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. Since then, several component models and their implementations have been proposed in the scientific community. The next section provides a brief list of component models.

3.3.2 Projects

3.3.2.1 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, and the semantic of bindings is a classical service call. Obviously, port types have to be the same. 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 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.

Typing of ports can act against the interoperability property. Bindings have a clear semantic, but can not use other communication link 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 it can not be easily changed at runtime.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	-	+	+	-	-

3.3.2.2 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 the system configuration. It means that component developers can not make any assumption about the context of use of each component, and designers can not change component behaviours.
Components do require services, provided by other component's typed ports.
A configuration describes an assembly of components. It handles the model of the application. To help in describing the system assembly, they propose compound components in which are described instances to be deployed, and their interactions(bindings). In this case, an action on a port of the compound component may have to be forwarded to internal components (eg: for initialization). To get rid of the ordering problem, they introduced *Modules* to handle one-to-many, many-to-one or many-to-many bindings. They are in charge of the propagation and have a pre-defined treatment.

The clear separation between the components development and the assembly creation (for a particular application) is a key of 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 may cause problems in future evolutions, in terms of interoperability. They introduced modules to handle specific communications between components and act like a proxy. They could have gone a bit further, and systematically use modules to specify how each connection. This information could help in solving issues, or specifying the system mor precisely.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	-	-	+	+	-

3.3.2.3 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 a content, which can be either a user class or other components in composite components. The model makes explicit the bindings between the interfaces provided or required by these components, and 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 simple, 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. There is no means to preview the effect of a reconfiguration, before actually executing it. No means to execute what-if scenarios to evaluate different possible configurations, etc. This lack of an explicit and independent reflection model, imposes most of the verifications to be realized 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 as Fractal are a bit opaque with respect to the outside world, making the openness and reuse by third party applications complicated if not foreseen in advance. Lastly, the dynamicity of an application running over Fractal is compromised, because the deployment of new components can not be done at runtime without restarting.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	-	+	-	-	+

3.3.2.4 uMiddle

Nakazawa et al are proposing a framework that bridges remote smart spaces called D-uMiddle in[NT07]. It makes it possible for a device to interact with another, over the Internet. This is made available by four distinct features of D-uMiddle. First, a local mapper mechanism abstracts sensor nodes into common representations. Second, a mechanism translates data transmission protocols from a node-specific one to a D-uMiddle common one. Third, a remote mapper mechanism creates proxies of sensor nodes from remote smart spaces in the local space. Fourth, a transport module enabling 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 burdening 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 equipments is thus made possible for free. Nevertheless, it does not supply tools to handle variability, adaptation or evolution of a deployed system.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	-	-	-	-	+

In the home automation domain, questions about space repartition of elements, remote communications or use are recurrent. It is sometimes complicated to spread a component based application over several execution platforms, because the communication links between components must be adapted.

3.4 (Web)Service Oriented Architectures

3.4.1 Description

Service oriented architecture is a paradigm, an idea driving the way to develop software. Software services are rendering services to other services or individuals. If an application, or a software, uses services to achieve its goals, it is a service-based software.

SOA vs. WSOA

In the Software Services community, service-oriented architecture is often used in place 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 a service-oriented software can use this service as any other. In conclusion, I would say that a service-based application is not necessarily using 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 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 1.3.1 illustrates this mechanism.

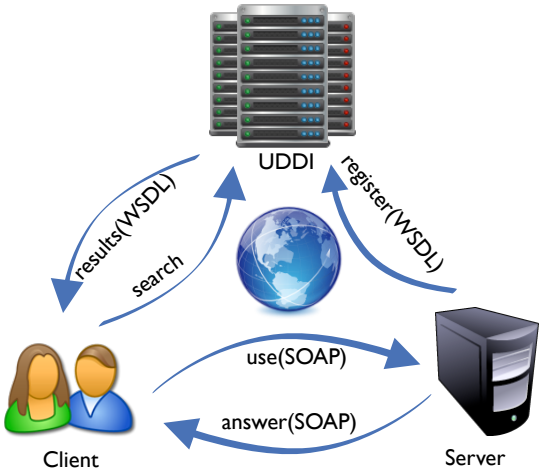


Figure 1.3.1: WebService Architecture

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 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.

If this approach provides mechanisms for dynamic search and use with precise descriptions, the amount data exchanged and the complexity of the SOAP messages 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[Fie00], Roy Fielding introduced in 2000 the term and idea of Representational State Transfer(REST).
REST has been designed to lighten transfers of information through web-based communications. In place of heavy serializations of concrete program objects, REST architectures are sharing representations of these objects using XML for instance. These representations are handling 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. **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 are handling information about the resource concerned by a request. For instance, a GET request on an URL like *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. Last example, a DELETE request on *http://myMedia.org/books/2517* should remove the book which 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 apparition of these two paradigms, ideas emerged about connecting software systems and everyday life objects. The presence of registries, or auto-discovery mechanism also lead to an abstraction of the physical location details. These interconnections finally brought new paradigms known as Internet of *.

3.4.1.1 Internet Of * and the Cloud

Researchers and industrial companies are done with Internet. Now, they want to investigate other kinds of Internets. More than sharing information, and facilitating interactions between people, researchers and solutions are today focussing on services through Internet. From this new breath was born the Internet Of Things (in which devices are offering services to the world) and Internet Of Services (where services are rendering services to other services, and sometimes, to human people).

Internet Of Things
Issued from the evolution of electronic devices, the Internet Of Things (IoT) relates to an approach in which objects of the 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, whith a stronger added value. Still in its infancy, the IoT is looking for software tools to develop and describe these interactions, as long as the services rendered.
Software components are quite convenient to virtualize everyday life objects, and service-oriented computing is convenient to describe and implement 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 produce promising results.

Internet Of Services

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

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. May 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 get the hardware configuration you need, just-in-time, to run your software as a service. You will never know where your software is really executed, but it will run in the cloud.

In this global context, the desire of a part of European research community is to lead the deployment and democratization of these ideas. To reach this objective, several computer science research laboratories join together in a Network of Excellence for Software, Services and Systems.

3.4.1.2 The S-Cube Network of Excellence

S-Cube[18] is a European Network of Excellence(NoE) in Software, Services and Systems(S^3). This NoE aims at making European research the leader in software-services revolution. By connecting research to industry, and unifying multidisciplinary researches, S-Cube tends to develop agile and holistic service engineering methods, and specify principles and techniques of service adaptation.
This European NoE has been funded by the European FP7 'Coordination' programme research under the ICT

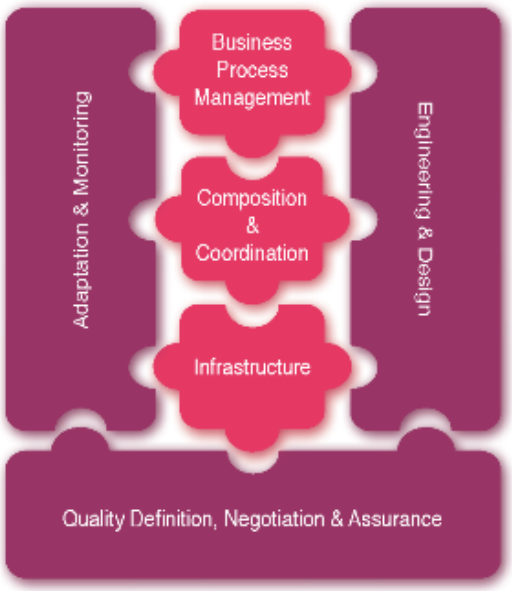


Figure 1.3.2: S-Cube Research Framework

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

theme. Along with strong collaboration and mobility opportunities beyond European research centres, S-Cube funded several PhD thesis in different layers of the S-Cube "BigPicture" (fig. 1.3.2).

Triskell skills of excellence 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 brought funding for two PhD thesis. The research leading to the results presented in this thesis has received credits from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Several tools have emerged to support these new ideas and paradigms for software developments. The following section details some existing projects and frameworks in this domain.

3.4.2 Projects

3.4.2.1 JBI

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 Enterprise Service Bus (ESB) as a basis to define a component model.
ESB refers to a business middleware family for service-oriented applications. These middlewares act 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. They are declared as any other service, within the scope of the ESB runtime.
The JBI component model has been designed to reuse Java technologies like WebServices, BPEL or JMS, and thus, avoid new specific developments. The specifications have been successfully implemented in several frameworks such as OpenESB by SunMicrosystem, ServiceMix by Apache Foundation, or PeTaLs by OW2. In JBI, components have independent life-cycle, and communicate through their services over a normalized message middleware. Actually, this middleware acts as an abstraction layer for communications, and eases the integration. JBI components are split in 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 can not 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 by the framework like this: service descriptions are encapsulated into Service Units, which are then encapsulated into deployable

business component 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 the 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 inter-dependencies, makes it impossible to dynamically replace components and/or reason about the system state.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	+	-	+	-	-

3.4.2.2 OSGi

OSGi is the Open Service Gateway Initiative. This association created in 1999 aimed at providing facilities for software integration and development. To achieve this task, the association released a set of specifications defining what should do any runtime implementation to reach a given service level. These levels specify which minimum set of service a runtime must offer to be compliant with the level.
In OSGi, services are given and contained in units of deployments 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 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. Even worse, 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. However, in our context, high dynamicity and adaptation skills are required. Moreover, there are very few reflection primitives and thus, interactions between bundles can hardly be traced or even made explicit.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	+	-	+	-	+

3.4.2.3 SOPRANO

SOPRANO (Service-oriented programmable smart environments for Older Europeans) was an Integrated European Project, which successfully ended in April 2010. Their 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. They integrated in their framework a context manager, able to give a virtual view of all devices, a process manager in charge of taking 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 the needs, and able to adapt to changes in the context. However, no attention is paid to the variability management, remote control or interoperability of devices.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	-	+	+	-	-

By nature, Service-Oriented software are highly dynamic and their architecture is not always easy to figure out. Indeed, the services used and the connection between software elements are never known prior to the execution because of the search and use principle. Component models for SOA have been invented to make the description of this kind of application more explicit. They also merge well known software component techniques with new services ideas, and conciliate the best of each approach.

3.5 Component Models for SOA

3.5.1 Description

Components, as defined by component models, are providing 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 are offering services, new component models have been designed to merge both paradigms. This section presents some famous implementations of these component models.

3.5.2 Projects

3.5.2.1 SCA

Service Component Architecture (SCA) [19] provides a component model for both the composition of services, and for the creation of service components. SCA is a model

19. http://www.osoa.org/

that aims to encompass different programming languages, frameworks and environments commonly used to build components and services, 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 the 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++ or the Spring Framework, are offering simple means to package a service or access any SCA service. For 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 accede to the component using Web services, JMS, JCA and EJBs natively. Here again, properties of services are described outside the code, to make changes much easier.

Policy Framework aims at offering means for the definition of security, authentication, quality of service, and other important policies of a service. Actually, the SCA Policy Framework makes use of the WS-Policy and WS-PolicyFramework 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 kind of system should behave. Thus, it imposes implementations to propose mechanisms for openness, evolution or remote control.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	+	-	+	-	+

FraSCAti
FraSCAti is an implementation of the SCA specifications. It is certainly the closest ap-

proach to what is required. Components can be composed into composite component. Communications between components are made using services, and can use several communication media. These elements make FraSCAti a serious candidate to address openness and remote control concerns.
In the last months, efforts have been spent on integrating FraSCAti and OSGi, which improved its faculties of evolution. In terms of interoperability, FraSCAti offers mechanisms for the connection of services, but do not address directly 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, a ad-hoc connector has to be created.
The FraSCAti script tool enables reconfigurations, adaptations of component assemblies. But the adaptations are limited to the manipulation of binding and component instances, which 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 issued from the SPL. Thus, features of FraSCAti can be deployed on-demand. However, this variability management concerns only internal features of FraSCAti, and an external tool still required to handle the variability of applications made with FraSCAti.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	+	+	+	-	+

3.5.2.2 AutoHome & iPOJO

iPOJO[EHL07] is the Apache service-oriented component runtime built on top of OSGi SOA platforms. 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.
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 development of the application itself, from autonomic management components. They tend 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: a middleware offering autonomic service-oriented component and a context facility, a runtime that includes monitoring and reconfiguration abilities, a set of service-oriented applications which represent per-

vasive components to be autonomously managed, and a set of managers organized in a hierarchy.

AutoHome, and the underlying iPojo component model, focus in giving management facilities for pervasive applications based on component models, and a service oriented runtime. This approach makes it possible to include in the application, 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.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
-	-	+	+	-	-

3.5.2.3 Gaïa Framework

Gaïa[RHC+02] is presented as a meta-operating system for ubiquitous computing, built on top of classical operating system. Its goal is to abstract 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 a 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 ComponentManagementCore 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, and Openness of the solution are not targeted by this work.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	-	+	+	-	+

3.5.2.4 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 behaviour and data gathered from several devices, enables the implementation of seamless, Internet-connected, web-based systems. And this, 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 shame the foundation for building applications to manage and control the devices.

Elevating devices access and control, at the level of services available through Internet, promotes interoperability since services are standardized. It also natively allows for the remote control of physical devices. According to adaptation, evolution or variability management of such kind of application, Niagara does not seem to bring interesting support.

Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
+	+	-	-	-	+

Chapter 4

Synthesis

The 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 interoperability of devices, but clearly lack of descriptions of the running application once deployed. They bring essential ideas to properly handle 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 representative of home automation devices. However, the specialized interfaces used as descriptions for ports, may prevent the realization of unpredicted connections.
Components for SOA is certainly the best approach for our concerns. Bridging components and services makes the benefits balance the drawbacks of each other.
Transversally, model driven engineering methods and techniques come with a lot of tools for virtual elements manipulations. They seem handy for runtime management of devices, for the description of software systems and for the variability management.

Table 4.1 summarize the good points and the lacks of the approaches described in the state of the art.

4.1 Good properties identified

Along the way through existing approaches, some good properties of design 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 get, and keep synchronized, an explicit and independent model reflecting the architecture living at runtime. This model makes

		Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
MDE /DSL	Habitation	+	-	-	-	+	-
	Dia Suite	+	+	-	-	-	+
	PervML	+	+	+	+	+	-
Component Models	Darwin	-	-	+	+	-	-
	Koala	-	-	-	+	+	-
	Fractal	-	-	+	-	-	+
	uMiddle	-	-	-	-	-	+
Service-Oriented Framework	Hydra	+	+	+	-	-	+
	JBI	+	+	-	+	-	-
	OSGi	-	-	+	+	-	-
	SOPRANO	-	-	+	+	-	+
Component Models for SOA	SCA	-	+	-	+	-	+
	FraSCAti	+	+	+	+	-	+
	iPOJO	-	-	+	+	-	-
	Gaïa	+	-	+	+	-	+
	Niagara	+	+	-	-	-	-

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 because of the 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 models.

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 means of description for these interactions. A clear and explicit description of the relations between components, gives a better understanding, and makes the analyse 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 description of interactions and dependencies are enforcing the independence of elements composing the system. It also improves the flexibility of the system, thanks to the possibility of modification 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 thus, support dynamic deployments and adaptations of the application during runtime with no restart. This is a basic facility offered by SOA execution environments.

Close Isolation enforcement
Component Models promote the close isolation 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 computation 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 interfaces descriptions and registries to expose the services to the world.

4.2 Points of contribution

In the electronic domain, the number of components and their always-possible connectivity offered technicians, and engineers, means to create various solutions. Even many years after their assembly, electronic devices can still be repaired or completed with new features. The proposition 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 by three aspects :

- A new component model that improves flexibility of software systems, by offering means for connecting any component to any other. This aspect addresses issues from interoperability 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 long 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 requirements of adaptation and evolutions at runtime, and validate the proposition

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 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 sucha tool.
This part presents the achievements of this thesis. While the first chapter gives information about some leitmotivs that animated this work, the remainder of the part details the contribution.

Chapter 5

Contribution

5.1 Global ideas

All along this thesis work, some recurrent ideas drove both the research of solutions and the development of the proof of concept.

5.1.1 Get 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 get rid of these constraints. This thesis strongly inspired of this domain to come to a solution. The link between electronics' solutions, and the contribution of this thesis, is stressed all along this part.

5.1.2 Make it possible

Scientific discoveries, and advances, are often due to hazardous reactions, unpredicted situations, and even due to errors. Software engineering tools of today, limit runtime failures by cutting down design elements to a set in which 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 validations highly customizable. Researchers are thus allowed to loosen checkers, in order to experiment new behaviors. They can make mistakes, experiment failures and adapt checkers with specific rules for their concerns.

5.1.3 Keep 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 the beginning to the end, solutions brought 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 engineers and technicians community, which require some tools to ease their work. Secondly is the system users population, like 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 details, 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 on figure 2.5.1, which provides an overview of the contribution.

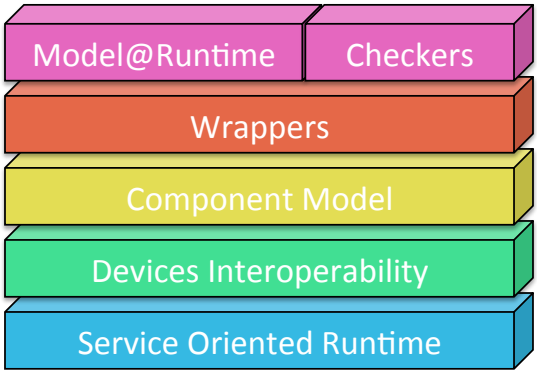


Figure 2.5.1: Overview of the EnTiMid layers

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

Component Model brings up structures and methods, to handle abstract representations of real devices. It provides a unified description of available ports, parameters, 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.

Model@Runtime & Checkers layer entails necessary tools to ease the management of the system. 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 the variability management. Checkers offers tools for validation, and improvements of the safety of the solution.

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.

Service Oriented Runtime comes to complete 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 the answer to 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.

	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 Britany "En Ti", which means "In house", and "Mid", for Middleware. It is thus, the middleware in the house. The component model has been realized using classical modeling techniques. Tools have been created to enable all functionalities.
Chapter 6 details each layer of this contribution.

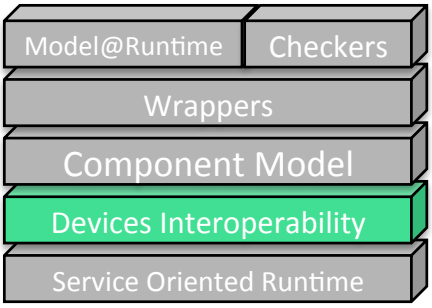
Chapter 6

Details on strata

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

6.1 Devices 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 are not able to communicate. Interoperability of devices is a central concern. It brings 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 impossible any direct interaction of devices coming from different brands. To overcome this barrier, some manufacturers worked in a consortium to define a unique communication protocol, for their respective products to be compatible. However, some of their products are coding boolean values on a single bit, while others are coding it on a byte. Again, two products may not be operable one to each other.
In [BRLM09] Bromberg et al. propose to automatically generate gateways between protocols, 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 an abstraction model of higher level.

6.1.1 Use of drivers

To realise this abstraction, drivers has been developed. A driver makes the link between, the 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. Naturally, they translate and transmit orders sent to a virtual device, to the physical device on the other way.
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' implementations 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 was created. A survey of devices' functions allowed us to extract the minimum set of common methods for each functions, as presented in figure 2.6.1 for instance. Once the set defined, a library containing all interfaces was compiled and included in the framework.
First experiments were promising. Interoperability was almost solved, but several

Figure 2.6.1: Functional Interfaces

drawbacks were rapidly identified while using this approach in real use cases. The set of interface is only extensible by augmentation of the framework. The development of a device driver could have been broken 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, an 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 to a shutter, the operation is simply not

feasible without an ad-hoc adapter (illustrated in figure 2.6.1). Interoperability was not solved.

6.1.3 Event-based approach

Since real-life events can not be predicted, we made use of event-base mechanisms. Message-Oriented Middleware (MOM) offers a simple, and efficient means of communications, using the producer/consumer principle. Consumers subscribe to a topic they are interested in, and producers just have to publish on the right topic. Thus, producers do not care about the presence of consumers. It is a good point to get rid of apparition issues.
To be able to use this approach, physical devices have been considered through two perspectives. *Sensors* are sensing 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* are acting on real life, using real-life equipments. They realize orders such as switching on the lights, or moving up 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 lit of actions that can be realized on the device. If a light can answer [*on,off*], a shutter would answer[*up,down,stop*]. For each action, actuators are waiting for messages on a specific topic. For a sensor to ask for an action to be realized, it must know the matching 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 are maintaining 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 realized. When an action is sensed, each message stored for this action is sent on its topic.

6.1.4 Example

For instance, figure 2.6.2 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. 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 get rid of 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 realized, 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.

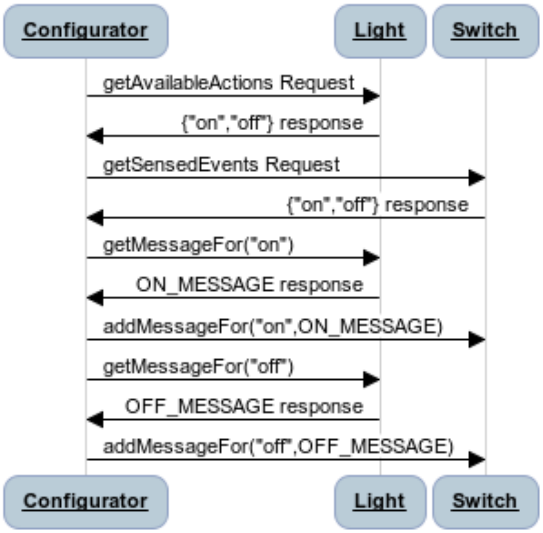


Figure 2.6.2: Configuration Phase

Let us consider a home with a switch to trigger a departure scenario. This scenario switches off all lights, and pull down shutters. In the considered example, there are only two shutters, and one light. The first shutter has been motorized when the house was built, and works on a KNX network, just as the scenario switch. The second shutter has been added after, and as 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 equipments. All these elements are visible on the left of figure 2.6.3.
The interoperation of all these elements is described in the case of a departure scene execution. 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.
4- Its activation makes 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.

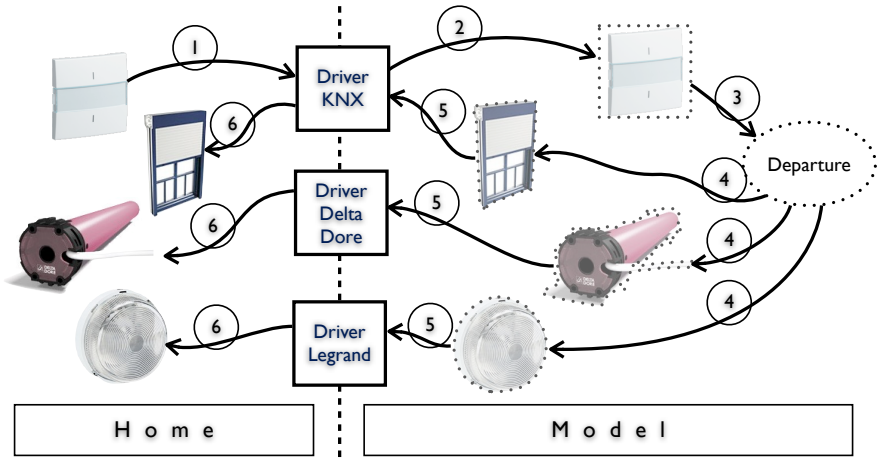


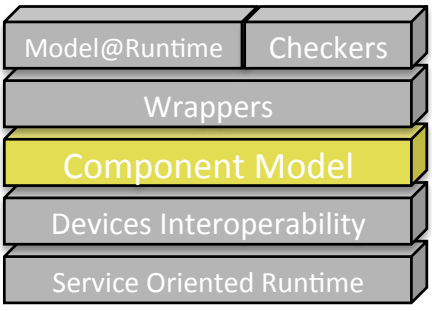
Figure 2.6.3: Example Interoperability

6.1.5 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 actions lists at design time. Each device provides information about the actions it supports, but a method has to be called. Since method calls can not be realized at design time, the only way to get available actions is to go and seek them in the implementation code.
The sequence diagram, presented in figure 2.6.2, describes a part of the work that a piece of program has to execute to set up the application's behavior. The sequence can not be implemented once for all, since the application may have to be adapted and to evolve while running. The configuration has to be expressed another way, to be able to change it, 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 the design of devices' assemblies. 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 explicit devices abilities and their interactions. 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 for real life devices, since they can use or provide services. Their interfaces(as list

of actions used or provided) are made explicit.
Their life cycle is very helpful to catch the dynamicity of devices' presence. Finally, the concept of components in software engineering is very close to electronic components. It makes it easy to understand and manipulate for anyone familiar with electronic components. Device manufacturers and software developers have here a common discussion base, a common language.
As presented in the state of art, component models are often too strict, and prevent from connecting components with non-identical interfaces. This restriction could compromise the interoperability gained by the *Device Interoperability* level, whereas this layer just aims at simplifying the configuration and the management.

This section organises as follow. Section 6.2.1 emphasis the relation between the proposed component model and electronic components, relation illustrated in section 6.2.2. The mechanisms responsible for the synchronization of model and code are presented in section 6.2.3. Lastly, the section 6.2.4 describes how the *Device Interoperability* integrates with this layer.

6.2.1 Make software components closer to electronic components

Talking about components, electronic ones are probably the first kind of component to come in mind of a lot of people. An electronic component, as shown in the bottom left part of figure 2.6.4, is a black-box surrounded by pins. The shape of the pins is standard and allows components to be connected to any board. Obviously, 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 realize its job. In the same time, they can trigger the execution of an associated task. Typical examples are the A and B input values of a comparator.
Output Ports release the data resulting of the execution of a task. The C result value of a comparator, or the tick of a timer, are two illustrations of this kind of port.
Parameter Port are used to set specific values for an instance. It specializes 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 strongly inspired from electronic components. Indeed,

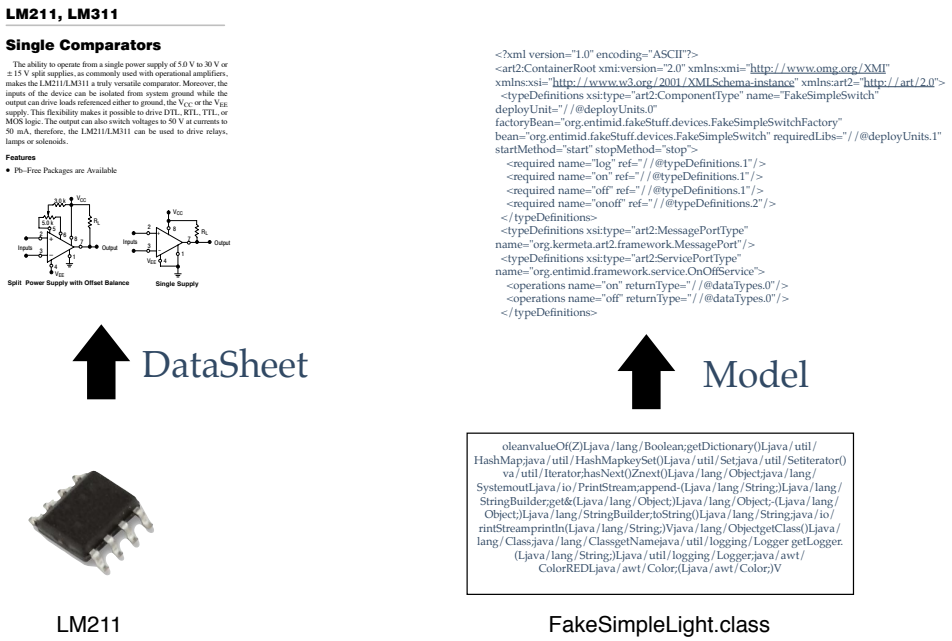
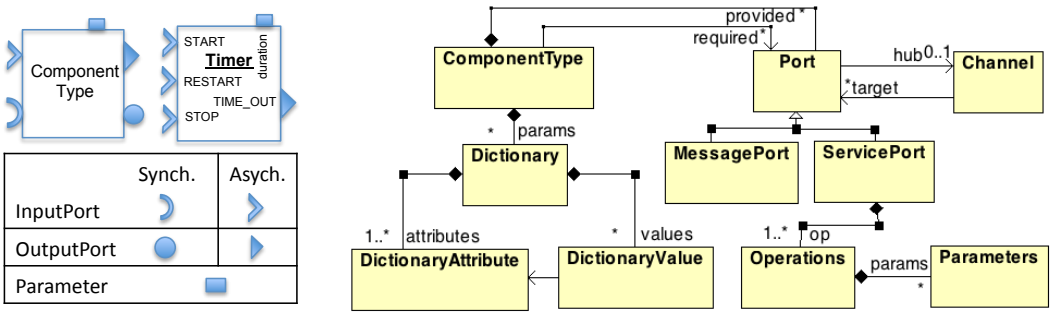


Figure 2.6.4: Electronic Parallel: Datasheets

InputPorts and *OutputPorts* have been implemented as presented in figure 2.6.5a. They have been divided into synchronous and asynchronous kinds, to handle both object-based method calls handling, and message-based communications between components. 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 of figure 2.6.5b. On their side *ParameterPorts* have been implemented as <key,value> dictionaries.

The following details each kind of port.

ParameterPorts This kind of port is used to specialize a 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 the 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. Types of both keys and values are Strings. This ensures the transmission of parameters in a unified way. Each component is responsible for the conversion from String to the real type of its parameters.



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

Figure 2.6.5: Extraction of a part of the component model architecture

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 2.6.5a 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 would have been defined in the StartStopRestart API.
Synchronous ports (also called ServicePorts in the model) are acting 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 components behavior. However, this is not the way of designing components 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 get rid of the implementation's typing, and deal with the typing at a higher level of abstraction.
Asynchronous ports, handled as MessagePorts in the component model(fig 2.6.5b), 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

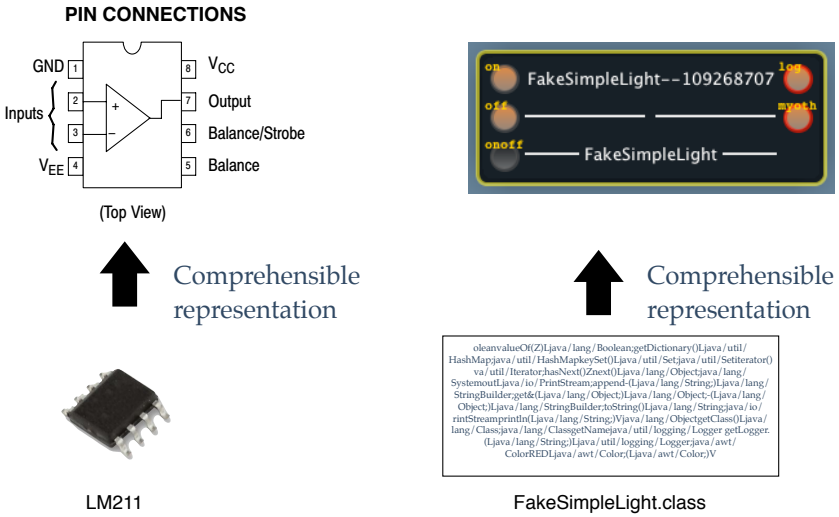


Figure 2.6.6: Electronic Parallel: Components

may change form a call to another. On figure 2.6.5, 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, on figure 2.6.5a 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 these, results in a parallel between electronic and software components as shown by figure 2.6.6.

6.2.2 Concrete example

This example shows how a real product is implemented in this component model. The RMG4S, on the left side of figure 2.6.7, 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(ie: 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 the dedicated switch), the state is propagated to any connected device, through

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

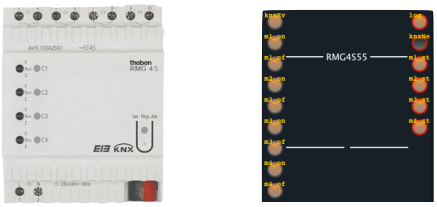


Figure 2.6.7: Example Model

the corresponding port of the component.
In addition, a *KnxEnv* input port allows the driver to circulate real-life events. On the other side, 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 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 way, 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 behaviour, and many other 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 has no effects on black-box components by the way).

6.2.3 Binaries and Model Relationship

The development of a component (ie: 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 the implementation code made by the developer. It is the *Code First*. These two approaches are not exclusive. The model of a component can evolve, and its implementation has to be impacted. Respectively, if a change is realized on 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, the listing 6.1 shows the complete implementation class

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(Object o) {
    if (timer != null) { timer.reset(); }
    getPortByName("logger", MessagePort.class).process("Timer("+time+")::STOP");
  }

  @Ports({ @Port(name = "restart"), @Port(name = "start") })
  public void restartTimer(Object o) {
    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");
  }
```

of a *Timer* component. This listing organizes as follow. 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 are rendering the services offered by the component. Life-cycle management methods are at the end of the class.

Component shape

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 realized 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 deploy units.
The last annotation just 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. 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 remove the port. Thanks to this multiple mapping, the user will be satisfied with no redundancy of code. Moreover, the transfer of the annotation from a 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 statefull 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

The meta-information, concerning the component model, are introduced in the code using annotations. This method for including meta-information in the code has already been used in tools like 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. *id est* the model is included as an XML file into the *.jar* that results from the compilation. The 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 experiment difficulties in placing all needed annotations to describe its component. The model first approach aims at providing tools to graphically (or textually) describe the component first, and ask for the generation of the implementation. This method is made available by the use of tools such as graphical DSL, textual DSL or generic meta-modeling languages like 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. Obviously, 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 a bit more complex. In fact, a temporary model is extracted from the existing code and an Abstract Syntax Tree (AST) of the code is created. The AST describes the code using a structured tree of object. 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 analysed, and modifications are made on the AST. The final code is generated from the modified AST.

The model first approach fairly helps in linking the model and the code. Anyway, 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 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.

6.2.4 Link with the interoperability layer

The actual implementation of the component model is a bit different from the view developers can have on components. The connection between two components is graphically as simple as drawing a line linking two ports (see the top of figure 2.6.8). Since ports can be synchronous or asynchronous, the runtime can not handle port connections in the same way. The activation of an output port implies different behaviors according to its kind. 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.

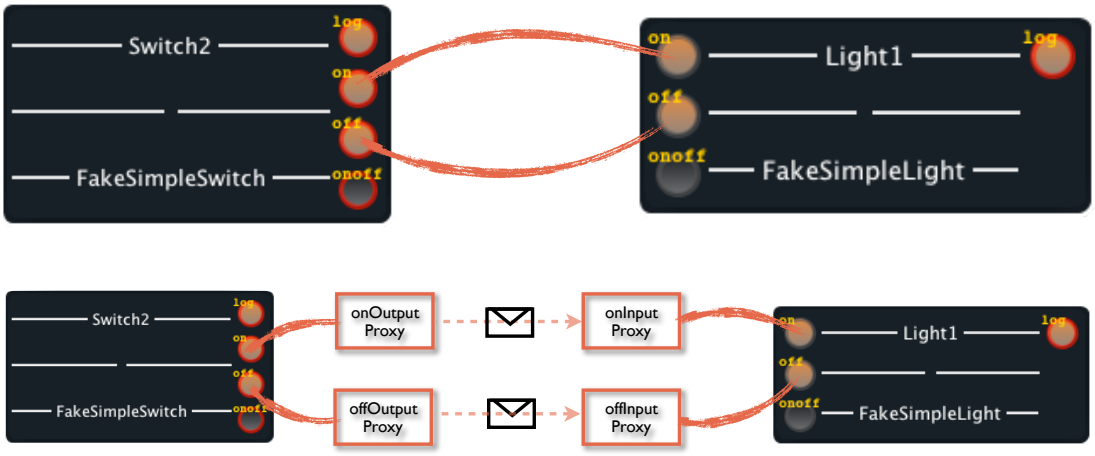


Figure 2.6.8: Link between interoperability layer and component connections

This complexity is hidden from developers, in both the code, and the model view of the components, by the use of proxies. At runtime, a proxy is generated for each port connected to another component's port, as illustrated in the bottom of figure 2.6.8.
If the port is a message port, the proxies are using 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 side, a proxy listens to this topic only, and activates the input port on which it is connected when a message arrives. Activations are realised 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 view: 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 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 case of distribution issues for instance), and enables some adaptation mechanisms.

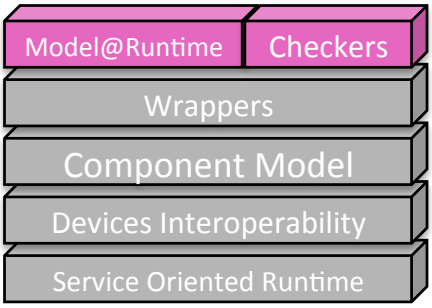
6.2.5 Summary

This new component model answers the need for a tool to explicit, at design time, the abilities of devices and their interactions. Annotations in the code simplify the integration of the component model into the implementation code, which 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 the configuration to be 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 any possible connection, 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 to assert their behavior. Assemblies have to be verified, and simulated, to prevent from any non desirable interactions.

To get rid of 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 proposition are making the *Model@Runtime and Reasoning Engine* layer.

6.3 Model@Runtime and Reasoning Engine



The *Component Model* layer provides a great 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 deployments. 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.

6.3.1 Check to validate

In electronics, the components assemblies have to be approved. Their conformance with relation to components, and applications specific constraints has to be guaranteed. This validation prevents assemblies from any computable damage. This conformance check is often realized by simulations, based on components' specifications described in their documentations. Figure 2.6.9 shows again, the parallel between the electronic approach and ours, where electronic simulations are replaced by model checking in our context.

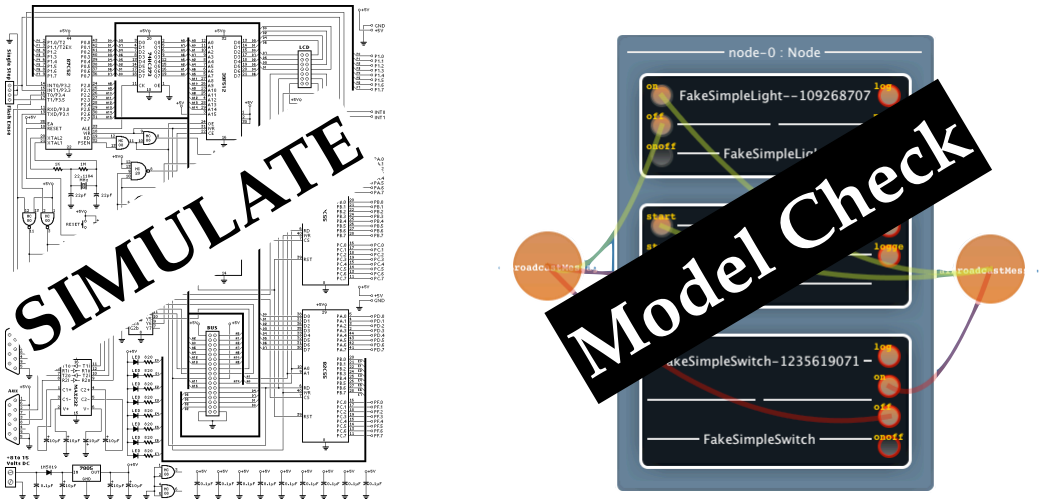


Figure 2.6.9: Electronic Parallel: Simulation

In a classical software engineering process, checking of conformance are made 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 in between design and deployment phases. Figure 2.6.10 displays the different moments where checks can be performed, and illustrates what can be check at each moment. These steps are detailed in the next paragraphs.

1. **Developer actions**
   The assembly tool can monitor developers' actions. During the conception, checkers can verify that only authorized operations are executed by the developer. When an inappropriate action is realized, the triggering of warnings and errors can improve the development process. Thanks to these 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.

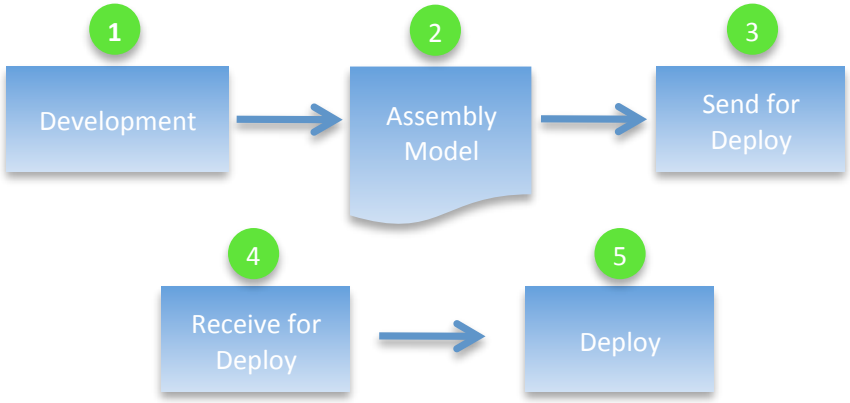


Figure 2.6.10: Checkpoints positions in the assembly deployment chain

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 aims at keeping a 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.
3. **Business Rules**
   An application created to control a plane has different constrains compared to a watering management system. Each application domain can require special rules to be 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.
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. Checks being 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.
   Modeling the platform resources could enable to move these checks before the deployment, and gain time.
5. **Check deployment commands**
   The last step of checking sits in the first step of Models@Runtime mechanisms. As explained in section 6.3.2, the deployment of a component assembly is split into

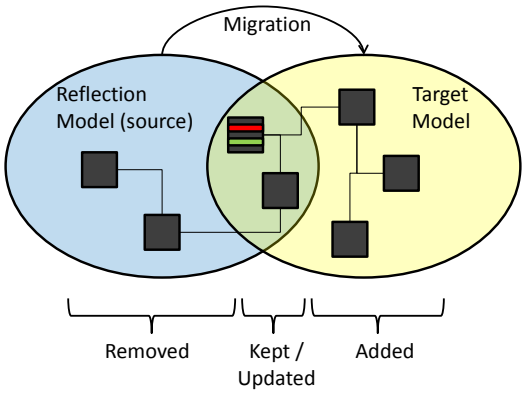


Figure 2.6.11: Identifying Differences between the source and the target configurations.

several commands. This last verification ensures all commands to be executable before running the sequence.

If the model of the assembly successfully passes all checkpoints, it is ready for deployment. 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) supervise the migration. This is explained in section 6.3.2.

6.3.2 The Model@Runtime engine work

Dr Morin has presented the Model@Runtime engine in his PhD Thesis[Mor10]. 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 (ie: identifies and sequence the necessary primitive commands). Lastly, it supervises the run of the migration commands 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.

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 2.6.11. 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.

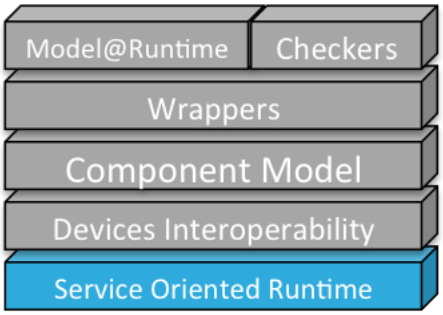
Plan the execution sequence
These commands are stored in a collection and ordered according to a heuristic[ADN+10] that ensures a safe migration from the current to the target configuration. Before actually executing the commands, the list is parsed to verify that all the commands can be executed. For example, for all *AddComponent* commands, the presence of the specific component factory is checked, to ensure all components can actually be added without problem. Doing this kind of verification for all commands ensures that the command execution will properly execute. 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 case of classic components, adding or removing bindings is as simple as setting or unsetting a variable. Generally, a component missing one mandatory binding is stopped, because it can not 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", 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 behavioural constraints can require more complex actions that 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 anywhere 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 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 offered all 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 good candidate to cope with 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 naturally imposes service-based applications to properly handle this requirement. Indeed, services can appear and disappear at any time, and applications built upon these principles obviously have to take these constraints into account.

The OSGi Alliance [osg], 'consortium of technology innovators', has released a set of specifications that define a service-oriented platform [All07], and its common services. This Service-Oriented runtime has been selected to support commands that require to add or remove 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 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 on 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 services updates, adds or removes. 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 deploy units and do not provide any service. They just embed components. When a **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 available, a new instance can be created.
Component instances are mapped on bundles too, because of their independent life-cycle management. Indeed, **start** or **stop component** commands are directly translated to start and stop bundle OSGi commands.

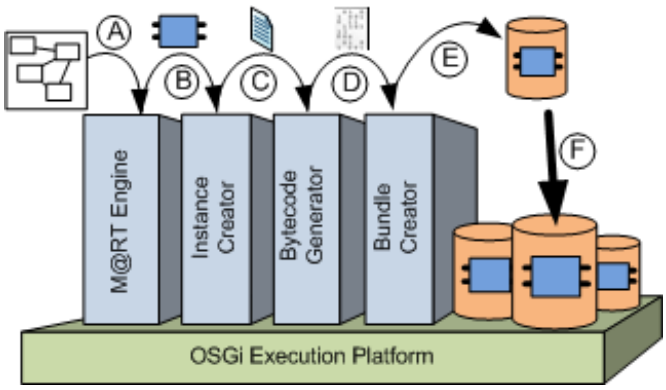


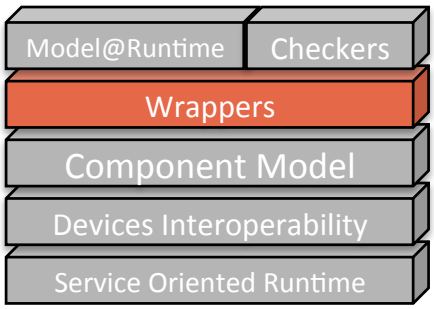
Figure 2.6.12: Instance creation tool chain

The creation of an instance is realized as presented on figure 2.6.12. 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 is taking 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 are often too heavy to be implanted into 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 get 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 the wrapper to 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. First, the export and uses 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 unregistrations 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 devices' drivers, a wrapper is required to be deployed for each application protocol to address. 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 handle communications to, and from the application level protocol.

6.6 Summary

The service-oriented architecture of the runtime makes it possible to cope with evolutions and adaptations, since bundles can be installed or updates with no need to restart the system. These facilities are exploited by the Model@Runtime layer, which comes with tools and methods to address 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 the flexibility, and enables connection of heterogeneous components, while keeping a high level of reliability thanks to checkers. Device Interoperability and Wrappers layers are providing respectively abstractions of manufacturers specificities, and free publications on application level protocols to promote interoperability and openness.

Part III

Validation

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

The contribution of this thesis has been validated by several elements.
Firstly, EnTiMid has been tested on a realistic use-case scenario, in which all properties previously listed have been stressed. This scenario, defined in collaboration with partners of an AAL project, is presented in chapter 7.
A second validation has been realized by bringing EnTiMid in front of a bundle of end-users, namely elderly people. This study had two goals. The first goal was the improvement of the Graphical User Interface (GUI) widgets presented to end-users, in order to make them as intuitive as possible. The second was the verification that people are able to act on, and get information from the home, according to a pre-define scenario. These two points are detailed in chapter 8.

Chapter 7

Validation in the context of an AAL project

As one of the IDA project's partners, we proposed EnTiMid as an integration tool, for the different equipments offered by industrials. Our initial wish was to come to an integrated, but very flexible, solution for elderly people. Even if this wish did not realize, EnTiMid had been considered and evaluated as a possible perspective for the future of AAL. This evaluation was based on a scenario, collaboratively defined with partners of the project. It has been designed to stress the properties required for such a kind of system.

The first section of this chapter introduces to the project and its context, and details the scenario. Sections 7.4 to 7.7 describe the evaluation of a set of properties: the environment setup, the procedures, and the results, for each property. Section 7.8 lists some threats to the validity of this study. The conclusion of these experiments is presented in section 7.9.

7.1 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 Rennes Metropolis(France) community. This local AAL project was funded by the Brittany region, to investigate on issues resulting from the ageing of the population, and its socio-economic impact. More precisely, IDA conducted 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 cares at home ASSAD du Pays de Rennes
Industrials Custos, Delta Dore, Urmet Captive, Spartime, i–Pocarte, Domtis, Ordimemo, Intervox Systems, Laudren, Solem
Social housing company Archipel Habitat

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

Installers Marsollier Domotique, Lepage Electronique
Project ownership assistance ARELIA
Research institutes INRIA, University of Rennes 1, University of Rennes 2, LOUSTIC, IETR, CRPCC
Public administrations Rennes Metropole, Conseil General d'Ille et Vilaine, Ville de Rennes, CCI Rennes, Critt Santé, MEITO
Elderly people Anonymous individuals for tests, obviously
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, life assistants, technicians, etc. The association helps more than 3,000 persons, split in 17 towns in the south and east of Rennes.

Although IDA was not funded by the European AAL programme, the assisted(elderly) person was project-centric, and the six dimensions described in section 2.1.2 appeared in watermark all along 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 this project was to measure, how ICT could foster the autonomy of elderly people at home, and support the activity of carers in this context. The idea was to offer an integration platform, able to deal with various devices and services, to promote the deployment of solutions adapted to each person's needs.
But rapidly, this idea was given up because every product tested in real homes, with real elderly persons, had to be left in the home. As a prototype of laboratory, EnTiMid could not offer sufficient guaranties of maintenance to be really deployed *in situ*.

Still interested by the proposition, a collaborative work had been engaged within the project. With the help of the ASSAD, Delta Dore, Custos, Arelia, and the LOUSTIC, for the most actives, a scenario has been defined to put EnTiMid in situation.
This case study was designed to be as close as possible to real life conditions. The story has been introduced by the ASSAD association. Products have been proposed by Delta Dore. The evolutions have been gathered from past experiences of the ASSAD, Custos and the LOUSTIC. This scenario was setup to evaluate EnTiMid on a realistic case, with the same devices as those actually deployed. It stresses several issues to measure how EnTiMid cope with them.
The following of this section presents the story, and the context of evaluation of EnTiMid. Issues are highlighted and for each, the way EnTiMid addressed the problems is described.

7.2 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 its family, and different

devices selected in collaboration with other partners, 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 to walk and move. To improve her safety, her daughter offered her to move to an equipped flat.
Among other equipments, the flat is basically equipped with an alert system, which triggers a voice call to a care centre 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 the care centre.

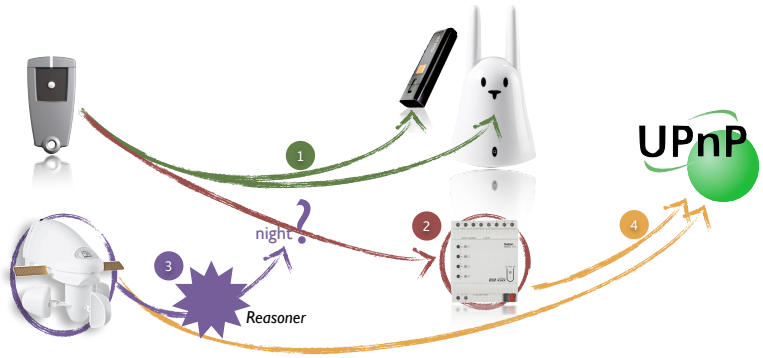


Figure 3.7.1: Solution elements for Mrs P.

The remote control, on the top left corner of figure 3.7.1, 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 catalogue. 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 7.4.

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 3.7.1, working on a KNX network. Section 7.5 presents how EnTiMid enabled this **evolution**.

Following this evolution, the behavior of the lights was sub-optimal, because lights

were switched on whatever the period of the day. To get rid of that issue, an **adaptation** mechanism(bullet 3), described in section 7.6, has been deployed. The sensing of daylight is realized by a KNX weather station outside the house. This weather station is visible on the bottom left corner of figure 3.7.1.

One day, her son came with a new device. This touch screen would offer Mrs P. to easily access to Internet, and have video calls with her children and grandchildren. The touch screen also has abilities for controlling devices over UPnP and DPWS networks. Since then, the deployed solution is required to be compatible and so, is required to export all available devices on UPnP and DPWS. This requirement stresses the need for **openness**. Section 7.7 elaborates about the mechanisms used to answer the fourth bullet of figure 3.7.1.

In order to evaluate the answers of EnTiMid in this scenario, and since a real deployment can not be realized, a test environment with real devices has been set up. Just before the description of the solutions offered by EnTiMid, the different elements composing this environment are presented in section 7.3.

7.3 Experimental setup

The test environment of this study has been realized with equipments provided by industrial partners of the IDA project for a part, and funded by the HID platform, financed by a European Found for Regional Developments, for the other part. It is mainly composed of two home mock-up, one with exclusively Delta Dore devices, the other with KNX devices only, and a MSI Top touch-screen PC, as visible in figure 3.7.2.

7.3.1 Delta Dore equipments

The Delta Dore mock-up, visible on the left side of picture 3.7.2, has been 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 thing on the table, down the mock-up).
- One *TYXIA 442* light dimming transceiver, and a *TYXIA 411* timed power switch, both hidden behind the rabbit.
- Not visible on the picture, two remote controls. One *TYXIA 110* with a single ON/OFF button, and a *TYXIA 141* with four.

Figure 3.7.2: Equipments available for the study

All these elements are using the X2D protocol, property 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.

7.3.2 KNX equipments

The second mock-up, on the right side of the picture, is made of KNX compatible products only, from mainly Theben manufacturer, and a bit of 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 enlightenment value. This information can be provided whether periodically, or when a value roughly changes.
- Just under the weather station, on the left, the *LUNA 113* is a light sensor for the outside. On its right, a *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 changing 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. A *RMG4s* device controls the four power sockets, at the very bottom of the panel, with a On/Off behavior only. Up 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 a IP connection. Helped by the Calimero [2] framework, it enables a programmatic control on all the devices, and allows to listen for event on the KNX network.

7.3.3 Other equipments

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

– A 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 on the picture, able to synthesize a voice from a text, and used to provide feedback to the user. A Nanoztag with a Mir:ror (grey little rabbit and blue circled base) are used as respectively a Radio Frequency IDentification (RFID) tag and an RFID reader.
– An Ethernet router to connect the KNX mock-up and the touch-screen

7.4 Interoperability issue

The first problem that the story of Mrs. P emphasis, is the connection of three heterogeneous devices.

7.4.1 Environment of the test

To evaluate EnTiMid on this question, we made use of the Tyxia 110 remote control from the Delta Dore mock-up, the 3G modem to send short 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 has been used for the deployment of the test.

7.4.2 Protocol of resolution

Drivers 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.

2. http://calimero.sourceforge.net

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 in the creation of a listener, and of a class to handle the implementation and 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.

Second element, the 3G modem has been considered as a simple modem. The sequence of *AT* commands to be sent to the modem, to send a short text message, has been collected from the modem documentation. With the help of a serial communication library in Java (RxTx), the component has been 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 in 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 a MP3 file. Indeed, the rabbit is able to whether, 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 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 3.7.3: 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 asks for 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 realizes the

required action.

7.4.3 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 for them to develop an ad-hoc device. This is not viable when targeting a provision of specialized solutions for each person's needs. EnTiMid makes this problem quite easy to solve.

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 context, because of their independence, and the avoidance of direct connections. For future applications, this signifies a great gain in time.

7.5 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 light are on at night.

7.5.1 Environment of the test

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.

7.5.2 Protocol of resolution

No KNX product had been used for the moment. So, the first task was to create the driver to control KNX equipments, and get a model representative for the RMG4s (presented in figure 2.6.7 a few pages backward). Once the component 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 enlightened when the Tyxia 110 button is pressed, in addition to the text sent, and the rabbit speech.

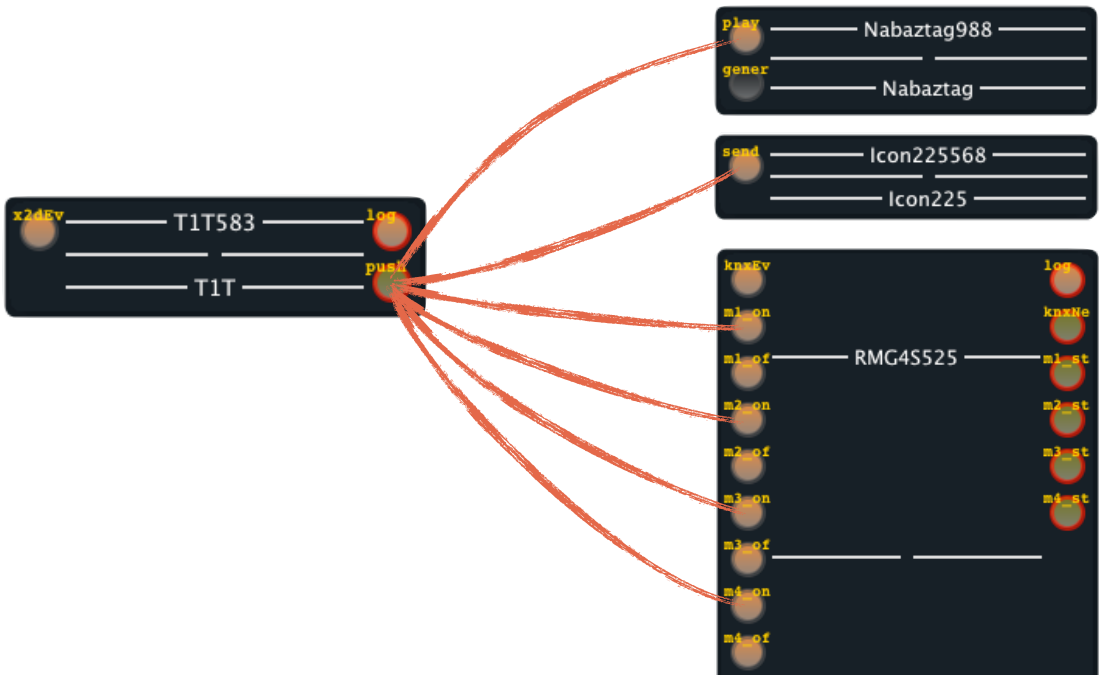


Figure 3.7.4: Components used in the evolution experiment

Last step, the technician sends back the model to the runtime of EnTiMid in the home. Once all checks passed, the runtime downloads the newly created component type and its driver, in order to create and connect the instance. All this procedure is transparent for Mrs P., who is just called before and after the operation to keep her informed. The OSGi runtime prevents from any service interruption.

7.5.3 Results

Again, a driver had to be created, and a component to handle the RMG4s too. 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. Realised 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.

7.6 Adaptation issue

The behavior of the lights was sub-optimal after the deployment of the evolution. Indeed, lights were switched on, whatever the period of the day, every time the remote control was pushed. To get rid of that issue, an adaptation mechanism(presented in

[NFM+10]) has been deployed. This mechanism changes the configuration of the system, according to the external enlightenment value sensed by a KNX outdoor weather station.

7.6.1 Environment of the test

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

7.6.2 Protocol of resolution

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 modification of) the running system configuration. The decision is taken 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 7.5.

To be able to perform execution time test benches, the experiment including the reasoner has been deployed from scratch. The MSI Top was cleaned of any previous binary element, a restarted. The deployment was then made as follow.

Initial Deployment The initial deployment is realised 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 rises again, the reasoner removes the connections and the instance of RMG4S.

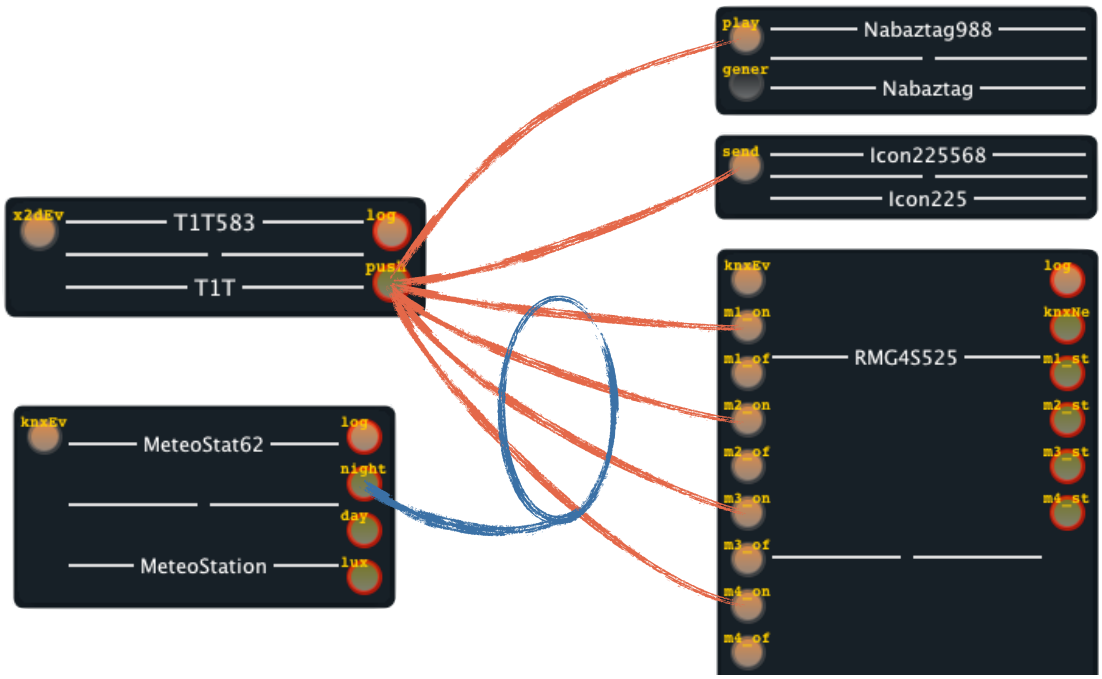


Figure 3.7.5: Components used in the adaptation experiment

7.6.3 Results

The figure 3.7.6 presents the execution times measured while a sequence of reconfigurations of the system was run. This sequence consisted in 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.

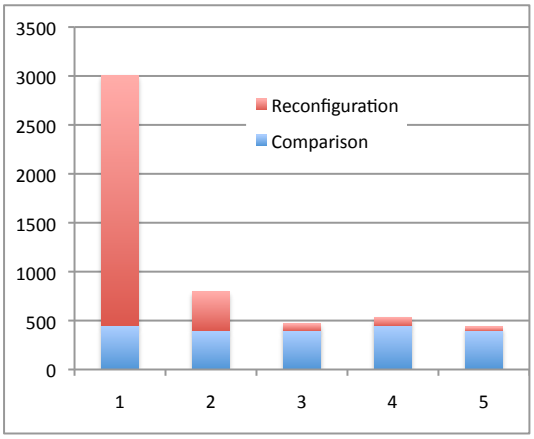


Figure 3.7.6: Time (in ms) spent in Configuration Comparison and Actual Reconfiguration

As the scenario has been considered in the worst case, 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.

7.7 Openness issue

As presented in section 7.2, 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].

7.7.1 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 3.7.7, organizes as follow.
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 is embeds.
Each service a device can offer has to be described in a separated file. This file characterizes all the UPnP-Actions the service render, and all the UPnP-StateVariables 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 parameters.

7.7.1.1 Environment of the test

This experiment required some devices to be deployed on the runtime for them to be exported. We choose 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, as the touch-screen brought by Mrs P.'s son. Since EnTiMid has been 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 only available for Windows.

7.7.1.2 Protocol of resolution

Mapping UPnP devices to EnTiMid devices
Not too far one to each other, EnTiMid devices and UPnP devices are not exactly aligned in their structures. However, the mapping (blue arrows in figure 3.7.7) has been quite easy to conceive. EnTiMid devices have naturally been mapped on UPnP devices. EnTiMid devices can provide two kinds of ports: services and messages ports. *NB: only input ports are considered here.* Services ports are composed of operations. This kind of ports has been associated to their 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, as many services as there are message ports are 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 for UPnP is thus made really intuitive and simple.
The services 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. Anyway, 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 which name is identical to the one of the port.

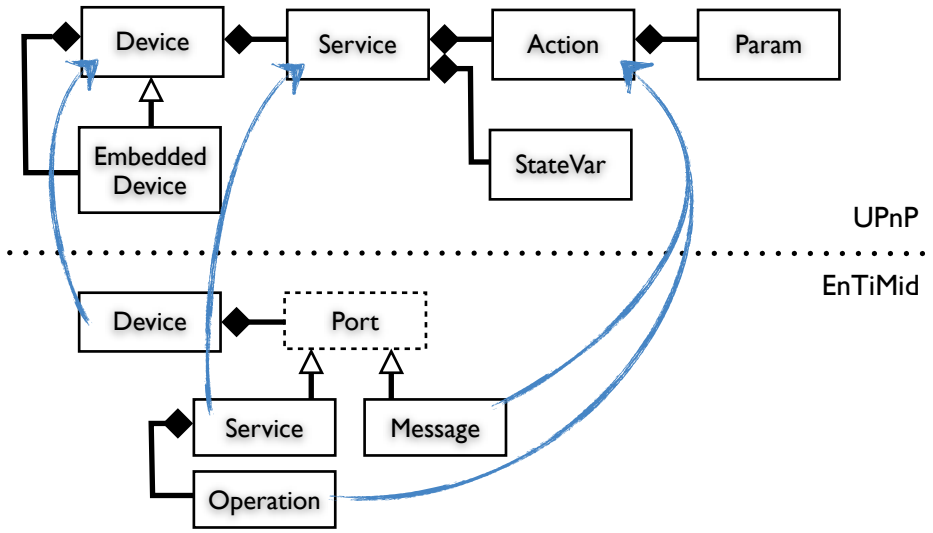


Figure 3.7.7: Mapping UPnP-EnTiMid

In a second time, 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 like 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 the sends in response the description files of all devices and services provided.

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 has been realized between the real runtime component standing for the devices, and the UPnP network. To cope with this question, 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. *ie*: an output port is especially 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.

7.7.1.3 Results

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

7.7.2 DPWS export

The 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 the UPnP one, described in figure 3.7.7. 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 virtual component is not sufficient in this case. Source code has to be generated.

7.7.2.1 Environment of the test

Since UPnP and DPWS are very close in terms of needs for devices to be exported, the same set of devices has been selected for this experiment.
As for UPnP, we made use of an external tool to check that the export of devices has been made properly. The experiment was realised using a second PC equipped with a DPWS explorer [3] to list and act on published devices.

7.7.2.2 Protocol of resolution

DPWS devices creation
For each device, service and operation, a Java class has to be generated. According to the

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

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 have been set up, and they are used at runtime to produce Java classes.
More than simple 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 has been 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 a quite 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.

7.7.2.3 Results

Obviously, this is not optimal but it works, since we were able to see all devices and act on them using the DPWS Explorer tool. Other solutions involving more powerful servers in charge of the transformation from a model representation to a component type bundle have been proposed, but not yet completely realized. Anyway, the principle is just to extract a tool chain that already works.

7.8 Threats to validity

7.8.1 Validity of the scenario

The scenario of experimentation, even defined in collaboration with several actors in the domain of AAL, 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 is possible to do with technologies, and industrials are not aware of every day problems the dependency of persons can pose, the discussions can rapidly come to a dead end.
The scenario validated for this study was accepted by every part, but may be limited by the comprehension each part had of the problem.

7.8.2 Variability management

The variability management, described as being an important problem 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 definition of a more global scenario may have stressed this requirement for variability management.
Nevertheless, several works have been realised to try to cope with that issue, such as an approach using Aspect Oriented Modeling presented in [MBNJ09]. Other perspectives to address this questions are presented in section 10.4.

7.8.3 Scalability

For the same reason, the scenario did not stress any question about distribution, and scalability of the solution for a deployment at the scale of a town, or even of a country. The contribution of this thesis, and its validation, could have been different if a real deployment had been required.
The MSI Wind Top, on which the experiments have been led, may not be the unique platform to test EnTiMid on, and a large-scale vision scenario would have highlighted this.

7.8.4 Safety and Security

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

About the safety, our experimentations did not require complex checks on models. Only simple structural checks, to find cycles for instance, have been implemented and used. Many checkers had not been completed since they were related to the business targeted. Their complete definition would have been useless in the context of the experiment. In case of real deployments, they have to be completed to verify that no configuration marked as failing is asked for deployment.

7.8.5 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 has been made possible by a 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 to act on devices through a web page interface. This product does not detect any event

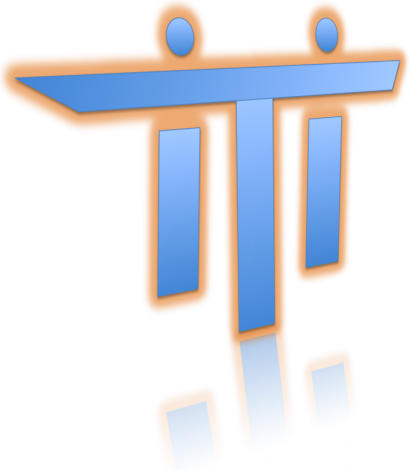
on the X2D network, it just acts on devices (with no acknowledgement by the way). EnTiMid is not able to use any device without means of communication with it.

7.9 Conclusion

The validation of the core elements of contribution of this thesis suffered from the lack of a real deployment. However, an experimentation using real devices has been set up to evaluate EnTiMid on a scenario as realistic as the collaborative work of project's partners could have made it.
EnTiMid successfully passed the main requirements stressed by the scenario, and required for such kind of systems to be deployed one day as integration platforms offering customized solutions for each person's needs.
Some limitations due to the lack of large scale deployment has been highlighted. These limitations will probably be addressed by the project of company in charge of the promotion of this technology in the industry.

Chapter 8

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 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.

8.1 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 obviously concerns some ergonomic aspects. However, the visual aspect was 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.

8.1.1 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 3.8.1 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.

8.1.2 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

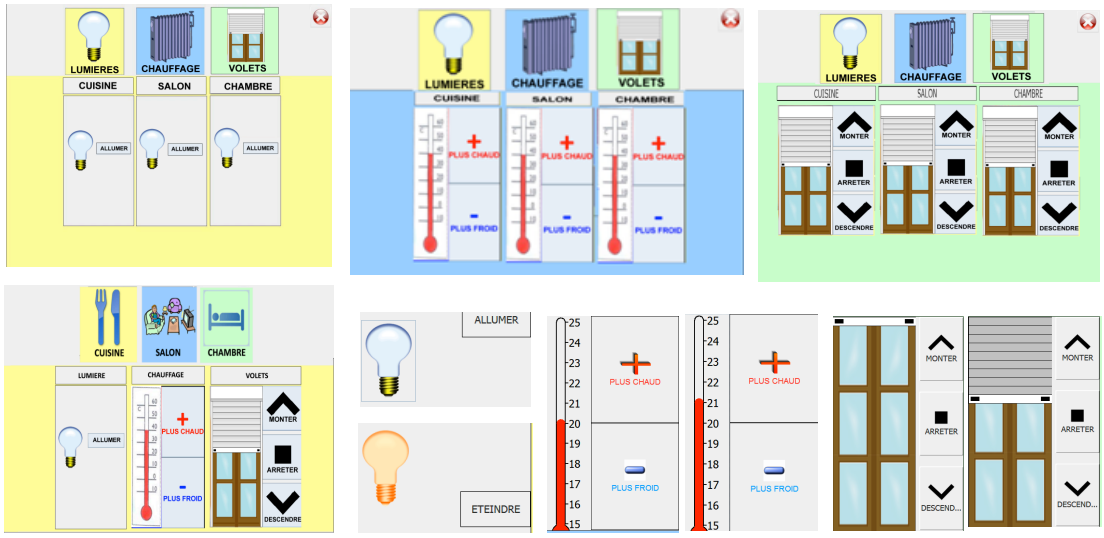


Figure 3.8.1: Results of the first phase

the bottom of figure 3.8.2. 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.

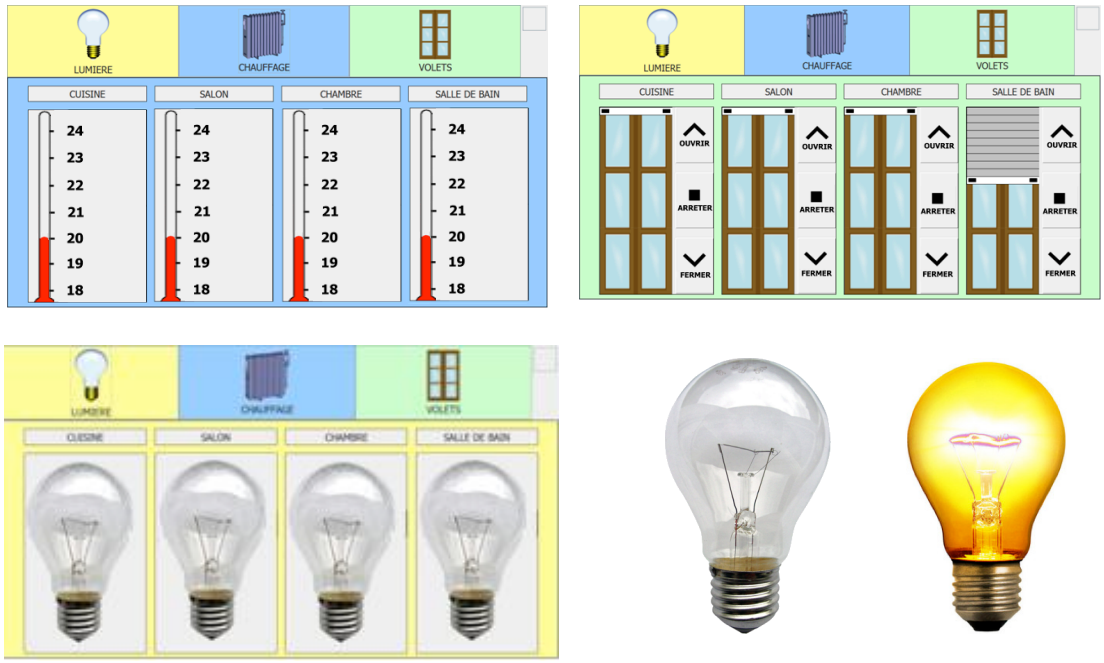


Figure 3.8.2: Improved interfaces, result of the second phase

8.1.3 Phase 3

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

8.2 Environment of tests

8.2.1 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.

8.2.2 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].

8.3 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.

8.4 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.

8.5 Results and conclusion

The table 8.1 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 8.1: 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, simply because of the improvement of the widgets.

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.

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 summarises by coming back on context and requirements, recalls the contribution, and emphasis its adequateness with relation to the requirements. After what, a short section discusses 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 synthesises the work realised for this thesis. Starting from the requirements, this chapter walks through the contribution, discusses its appropriateness to the context, and ends by highlighting some benefits and drawbacks.

9.1 Recall of context

The ageing of the European population incited the community to search for solutions to support this evolution. In this context, several issues have to be addressed in parallel. First, the domain of health care suffers from a manpower shortage, that could result in a decrease of the health service quality. Then, places in health care centres are not indefinitely extensive, and centres will shortly reach their maximum capacities. Finally, a day spent in health care centres, 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 kind of projects, and emphasis the interest of Europe in advances in this domain. The *Innovation Domicile Autonomie(IDA)* project, initiated by the metropolis of Rennes, 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 analysed to work out their possible contribution to this problem. Rapidly, the survey demonstrated that a unique solution can not be applied in all cases. Each person has different needs and requirements, which imply the solutions to be adapted for each deployment. Also, manufacturers reach their limits when a device has to be specialized for each user.

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 the 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. Anyway 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 all devices to be re-engineered to implement this communication interface. With this approach, all products already available can not 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 everyday life actions, have to take into account the environment in which they are executing. They should be able to dynamically adapt to changes while running, in order to maintain a level of services, or functionalities. Obviously, these adaptations should not require any restart of the system that would disable all functionalities for the time of restart.
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 the system to be accessible through a specific application protocol. This availability offer 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 to external contributions. Each one can take part in adding a smart behaviour, on top of a reliable set of functionalities.
The remote control may be required for a carer to remotely check if everything is doing well. It may also help in supporting a 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 to their needs, the set of options of the system may become hardly manageable. Some deployments may require high computa-

tion 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 even more complex the management of versions. All these elements lead to a huge number of configurations, and to a 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 tends to improve the quality of life of dependent people. A minimum service level has to be guaranteed, for inhabitants not to stay stuck in the house in case of emergency for instance. Moreover, accesses to the system have to be secured to disallow anonymous controls, but not annoying for authorized people in a normal use case.

Acceptability & Accessibility are issues that must be addressed, particularly when a software system takes responsibility for a 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 ensured to keep a hand on things that can happen.

9.3 Survey of existing approached

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

The 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 interoperability of devices, but clearly lack of descriptions of the running application once deployed. They also bring essential ideas to properly handle the sporadic apparition of elements, since a service can be started and stopped at any time.
Component-based architectures provide an ideal abstraction level to meet the requirements for a virtual representative 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.
Components for SOA is certainly the best approach for our concerns, since the benefits from the first balance the drawbacks of the other.
Transversally to any approach, model driven engineering methods and techniques, come

with a lot of tools for virtual elements manipulations. They seem handy for runtime management of devices, for the description of software systems, and for the variability management.

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

9.4 Outline of the contribution

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.

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 in having 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.

Model@Runtime & Checkers layer entails 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 in enabling management of the system at runtime, in offering tools for checks and validations, in improving the safety of the solution, and help in dealing with the variability of the system.

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 offers all devices to be available on application level protocols for free.

Service Oriented Runtime comes to complete 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 art, led to an extraction of a list of requirements. These requirements have been stressed as 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. The 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. The 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 had been identified. The contribution of this thesis is conservative with relation to these properties.

Reflexive Model proposed by MDE is still available. The model@runtime layer is responsible for this ability, and keeps synchronized an explicit and independent model reflecting the architecture living at runtime. This model allows for the creation of rea-

		Interoperability	Openness	Adaptation	Evolution	Variability Management	Remote Control
MDE /DSL	Habitation	+	-	-	-	+	-
	Dia SUite	+	+	-	-	-	+
Component Models	Darwin	-	-	+	+	-	-
	Koala	-	-	-	+	+	-
	Fractal	-	-	+	-	-	+
	uMiddle	-	-	-	-	-	+
Service-Oriented Framework	Hydra	+	+	+	-	-	+
	JBI	+	+	-	+	-	-
	OSGi	-	-	+	+	-	-
	SOPRANO	-	-	+	+	-	+
Component Models for SOA	SCA	-	+	-	+	-	+
	FraSCAti	+	+	+	+	-	+
	iPOJO	-	-	+	+	-	-
	Niagara	+	+	-	-	-	-
	EnTiMid	+	+	+	+	-	+

Table 9.1: Adequateness of the contribution

soners, able to perform changes on the model, with no immediate impact on the running system, until the model is conform.

Externalized coupling is provided by the model@runtime, by enforcing the elicitation of links between components, and by the presence of drivers, which can not presume of any use of the components. 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, is also taking 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 are making the development of components easier. 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, are also simplifying 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 in editors, and components are assembled and connected using drag-n-drop interactions. If checkers 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 device are already familiar with this way of connecting things, since the model is very close to the electronic components. The conception 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 evolutions 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 manufacturer, protocol used, or media used, but can also be due to the thing they represent. Several services available through the Internet have been wrapped into components, to enable the application to access to a service such as a on-line calendar, a weather service, a picture sharing service, and even a famous social networking service. The component model enables to seamlessly connect a Google Calendar from internet to a light component for instance. The effect of such a connection could be to switch off the light where no appointments are specified.

9.8 Limitations identified

9.8.1 Behavioral description

The component model eases the structural description of a software system, where people are much 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 as soon 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 the 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 solved at the model level, before the deployment.
If this problem has not been addressed in this thesis, it has already been identified and scientists already proposed some solutions under terms like component connectors, which can have complex behaviors (cf Beugnard et al. in [MB05]). Mechanisms such as renaming or mappings of parameters could be implemented to cope with this issue.

9.8.3 Too light checkers

Our experimentations did not require complex checks on models. Only simple structural checks, like cycles detections, have been implemented and used. Many checkers had not been completed since they were related to the business targeted. In case of real deployments, they have to be completed to verify that no configuration marked as failing is asked for deployment.
Realized 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, depend on what has to been checked. Since none had been completely implemented, the information currently available in the model can be too poor for checkers to work.

9.8.4 Variability management

The 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 by hand. 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 Not tested on 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 a 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. Anyway, the runtime of EnTiMid requires a Java virtual machine to be deployed on the platform first, and thus enough power for the JVM to run.

9.9 Contribution to the S-Cube NoE

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 [1].
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 1.3.2). Coupled with other layers, it can take part in the cross-layer adaptation mechanism.

1. DoW Amendment 4, December 6th, 2010

Chapter 10

Scientific perspectives

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

10.1 IDA, second phase

Conclusions from the experimentation led in the context of the IDA project shaped a promising future for the use of such kind of tools, in the AAL domain. Even if the maturity of EnTiMid, and the objective of the first phase of the project did not permit to test EnTiMid in a real deployment, the protagonists (industrials, carers, social workers and elderly people) have shown an interest for 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 the variability management, on real deployment.

10.2 End User Programming

The End-User Programming [AP10] 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 inhabitant must be able to keep control on things in their homes, end-user programming sounds like very promising, but yet challenging perspective.

10.2.1 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 probably would express their requirements for the behavior of the system in another way. The question is which one.

At our disposal, a range of description tool from a textual domain specific language, to a visual language composed of icons and boxes, linked with arrows. A solution for this problem probably stands somewhere in between these two extreme propositions, 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 to be unable to express himself as usually.

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 a failure. Secondly, people naturally express the nominal behavior, without taking care of 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 different kinship with technologies, systems should offer several edition tools, out of which the end-user selects the handiest for him.

10.2.2 Fuzy Logic and Learning algorithms

In the hypothesis people are able to describe a behavior of a function, how could they now about bound of values. In other words, if a user is defining a behavior for the light, how could he know the minimum and maximum values 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 simplify again the description of an application behavior.

10.3 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 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, as a firmware, to ease their integrations.
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.

10.4 Architecture Synthesis

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

10.4.1 Dynamic Software Product Lines for the management of variability

Not really addressed in the contribution, nor experimented in the validation, the management of the variability in the domain of AAL and Home Automation still is 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 the variability.

As proposed in [Mor10], Aspect Oriented Programming, coupled with Software Product Lines can be used to address this problem. Product Lines, well-known variability management tool for supply chains, has been transposed in software domain under the name of Software Product Line (SPL). Large scale productions like cars handle 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 huge number of variable elements, such as component-based applications, can be defined the in the same way. Base functionalities of the software are described in the base product, and specific options are plugged-in

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

according to the customer selection. The problem is that these tools have been set up to ease the one-shot creation of a products.

Dynamically adaptive software systems are able to dynamically evolve after their creation, and SPL 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 realized by Carlos Cetina et al. in this domain, presented in [CGFP09], is very close to what we want to achieve and reflects our future.

10.4.2 How to describe the behavior ?

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, nor 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 case of reconfiguration. In a word, the structure of the assembly. Nowhere the interactions between components can 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 a work in progress, this approach still has to be experimented in more depth.
Cassou et al. recently presented another approach to this problem of description of interactions. In [CBCL11], they introduce a "behavioral contract". These contracts aim 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 a future work.

10.5 Kevoree

EnTiMid, as an achievement of this thesis, has been marked 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 a customization for home automation and AAL, of a more generic tool. Its name is Kevoree [2]. The core mechanisms of adaptation,

2. http://kevoree.org

evolution, etc. have been moved in Kevoree in order to make them available for other use cases than home automation.

Now, if 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. Among them, 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 a work in progress in the context of another thesis, which explore 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 tries 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's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments, according to the manufacturer web page. Recent developments shaped the idea of using Kevoree and the component model to ease sketches, and deployment of applications on Arduino platforms.

10.6 Open Control/Command operating system

The component model of this contribution has been designed to allow for 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 with relation to real devices specificities, enable 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.

Chapter 11

Industrial perspectives

The contribution of this thesis interests several publics. The general public is curious to know about how computer science can help in improving the elderly people's quality of life at home, since EnTiMid was initially designed for the Ambient Assisted Living context. Everyone has or had a family member who could have been helped by such a proposition. Industrials are more interested by the good properties of the contribution. Since device manufacturers are familiar with electronic components, they easily understand how this component model works.
Starting from the validation scenario, a demonstration of EnTiMid has been set up and presented, in 3 years, in more than 10 public and scientific events. This demonstration stressed the adaptation aspect of the contribution in a AAL context.

11.1 Public demonstrations

Public demonstrations are an opportunity for the general public to discover what issues scientists are trying to solve, and how they do it. On the other hand, these meetings are a chance for scientists to collect feedback on their work, from naive persons. Naive, in the sense that they are thinking about the problem for the first time. The questions asked are often very interesting, incite to step back from details, and consider the proposition 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*[1] in French, is a national event lasting for 3 days. From Friday to Sunday included, scientist 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 secondary school children. 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

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

Figure 4.11.1: Science festival presentation

by all of them. This first day did not bring a lot of useful comments. At least, less than the two other days.

On Saturday and Sunday, the festival is open, for free, to families, and anybody interested by sciences. These two days were a hard test for EnTiMid, and full of interesting talks with people. Indeed, EnTiMid had to run from 8am to 8pm without failures, in spite of the touch screen stress due to children's fingers, and in spite of the unpredicted use cases asked in live by visitors.
It was the place for EnTiMid to get enriched by new ideas, remarks, or people experiences in facing elderly people's dependency problems. Never, for the all duration of the festival, somebody said 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 public. Concerned by the studies, questions of these two events' visitors were more precise, and more focussed about 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. It makes, in my opinion, a validation of 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].

11.2 Industrial perspectives

The success of EnTiMid has generated some industrial contacts. Several companies were interested by 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 to 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 development of EnTiMid, to make it ready to be used in the industry, had been submitted to the Brittany region, and accepted. This project funded the work of an engineer for a year, whose task was twofold. First, to redevelop some parts of the contribution for it to be more stable, and then, support the promotion of EnTiMid toward the industry. The development methods of EnTiMid have been clarified, and a second demonstration has been set up to focus more the core functionalities, and less the home automation / AAL aspect.

In addition, a project of company creation is currently in progress, and should end with a 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.

Acronyms

AAL Ambient Assisted Living. 16, 19, 23, 24, 89, 90, 108, 119, 121
ADL Architecture Description Language. 32
AST Abstract Syntax Tree. 73

DLNA Digital Living Network Alliance. 82
DPWS Device Profile for Web Services. 30, 82, 92, 100, 103
DSL Domain-Specific Language. 23, 31, 32, 34, 50, 73, 131
DSPL Dynamic Software Product Line. 130

ESB Enterprise Service Bus. 41

GUI Graphical User Interface. 87, 107, 110–112

HVAC Heating Ventilation Air-Conditioning. 18

ICT Information and Communication Technology. 23–25, 89, 90
IDA Innovation Domicile Autonomie. 89, 125, 127
IoS Internet Of Services. 40
IoT Internet Of Things. 39

JBI Java Business Integration. 41, 42

MDE Model Driven Engineering. 23, 31, 34, 49, 74, 121
MOM Message-Oriented Middleware. 63

PaaS Platform As A Service. 40
PLC Power Line Communication. 18, 26, 27, 29, 92

RFID Radio Frequency IDentification. 94
RPC Remote Procedure Call. 44

SaaS Software As A Service. 40
SCA Service Component Architecture. 43, 44

SIP Session Initiation Protocol. 30

SOA Service Oriented Application / Architecture. 23, 43, 45, 49, 51

SPL Software Product Line. 129, 130

UDDI Universal Description Discovery and Integration. 38

UPnP Universal Plug & Play. 29, 30, 82, 92, 100, 103

WSDL WebService Description Language. 38

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), Sydney, Australia, December 2010. short paper.

[All07] The OSGi Alliance. Osgi service platform core specification, release 4, April 2007.

[AP10] Tuan Nguyen Dao Anh and Heino Bøg Peter. End-user programming. Master's thesis, Aalborg University, June 2010.

[Arn] Terrell E. Arnold. Terrorism and the fear market.

[ASS10] ASSAD. Les travaux ida - phase 1. Technical report, IDA, Juin 2010.

[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, Cape Town South Africa, 05 2010.

[BCL+06] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.B. Stefani. The FRACTAL Component Model and its Support in Java. Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems, 36(11-12):1257–1284, 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.

[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] F. Chauvel, O. Barais, I. Borne, and J.-M. 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 ICSE'11: Proceedings of the 33rd International Conference on Software Engineering, pages 431–440, Honolulu, United States, 2011. ACM.

[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.

[CDT08] H. Cervantes, D. Donsez, and L. 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.

[Com] European Commission. Fp7 tomorrow's answers start today.

[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).

[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.

[Fie00] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. PhD thesis, 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.

[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.

[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, Grégory Nain, and Jean-Marc Jézéquel. Taming Dynamically Adaptive Systems with Models and Aspects. In 31st International Conference on Software Engineering (ICSE'09), Vancouver, Canada, May 2009.

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

[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, DOCTEUR DE L'UNIVERSITÉ DE 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), 2006.

[MSCP06] Javier Muñoz, Estefania Serral, Carlos Cetina, and Vicente Pelechano. Applying a model-driven method to the development of a pervasive meeting room, 2006.

[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 schizofrenic middleware for home/building automa-

tion. In In ServiceWave'08: Networked European Software & Services Initiative (NESSI) Conference, Madrid, Spain, December 2008.

[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] J. Nakazawa and H. Tokuda. A middleware framework for sharing sensor nodes among smart spaces. In Networked Sensing Systems, 2007. INSS '07. Fourth International Conference on, 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.

[osg] Osgi alliance. http://www.osgi.org/About/HomePage.

[otEU] Official Journal of the European Union. Decision n. 743/2008/ec of the european parliament and of the council.

[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. Pervasive Computing and Communications, IEEE International Conference on, 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 Frank Eliassen and Rüdiger Kapitza, editors, Distributed Applications and Interoperable Systems, 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.

[SML+10] A. Sixsmith, S. Müller, F. Lull, M. Klein, I. Bierhoff, S. Delaney, P. Byrne, R. Savage, and E. Avatangelo. 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.

[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.

[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, 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, 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

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.