RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

Génération de mandataire pour l'interopérabilité des services

Charbel EL KAED[1,2], Yves DENNEULIN[1], François Gaël OTTOGALLI[2]

Université de Grenoble[1] et Orange Labs[2].
nom.prénom@orange-ftgroup.com, nom.prénom@imag.fr

Résumé
Les protocoles *plug-and-play* couplés avec les architectures logicielles rendent nos maisons ubiquitaires. Les équipements domestiques qui supportent ces protocoles peuvent être détectés automatiquement, configurés et invoqués pour une tâche donnée. Actuellement, plusieurs protocoles coexistent dans la maison, mais les interactions entre les dispositifs ne peuvent pas être mises en action à moins que les appareils supportent le même protocole. En plus, les applications qui orchestrent ces dispositifs doivent connaître à l'avance les noms des services et dispositifs. Or, chaque protocole définit un profil standard par type d'appareil. Par conséquent, deux appareils ayant le même type et les mêmes fonctions mais qui supportent un protocole différent publient des interfaces qui sont souvent sémantiquement équivalentes mais syntaxiquement différentes. Ceci limite alors les applications à interagir avec un service similaire. Dans ce travail, nous présentons une méthode qui se base sur l'alignement d'ontologie et la génération automatique de mandataire pour parvenir à une adaptation dynamique de services.

Mots-clés : SOA, Plug-and-Play, Ontology, Génération de Code

---

1. Introduction

Les systèmes ubiquitaires pensés par Mark Weiser [30] sont devenus des réalités grâce à l'émergence de systèmes embarqués et des protocoles comme *plug-n-play* : UPnP [26], IGRS [16], DPWS [21]. Ces protocoles, se reposent sur les architectures orientées service, permettent la détection automatique de dispositifs et de leurs services associés dans un réseau domotique. Lorsque des dispositifs «plug and play» sont connectés au réseau local, des applications déployées sur des terminaux les découvrent et les contrôlent. Le but de ces applications est d'orchestrer les interactions entre les dispositifs et leurs services. Par exemple une application de partage de photos détecterait un appareil photo numérique et, sur ordre de l'utilisateur, afficherait les photos sur une télévision connectée puis imprimerait celles sélectionnées. La configuration de chaque dispositif, TV, imprimante, appareil photo, et leurs interactions seraient transparentes pour l'utilisateur qui aurait eu à installer l'application sur son terminal.
Chaque dispositif annonce les services qu'il fournit suivant un format et une syntaxe qui lui est propre. Par exemple une lampe UPnP propose le service `SwitchPower` avec l'action `Switch` (True/False) pour contrôler la lampe alors que le dispositif équivalent utilisant la norme DPWS propose l'opération sémantiquement équivalente `SetTarget`(On/OFF). Le problème est que l'hétérogénéité syntaxique de la description des services et la diversité des piles protocolaires de chaque dispositif empêchent une utilisation directe par les applications. Pour supporter les autres protocoles, les développeurs d'applications doivent implémenter toutes les interactions possibles avec les dispositifs équivalents en se reposant sur les différentes descriptions et fonctionnement standards ainsi que les couches protocolaires. Ceci est un travail laborieux et nécessite un temps de développement non négligeable.
Pour résoudre l'hétérogénéité, nous proposons d'utiliser les profils et descriptions standards UPnP comme pivot. Notre approche, consiste alors à générer des mandataires UPnP implémentant des descriptions UPnP standards pour représenter les autres dispositifs non-UPnP, voir figure 1. Ceci permet alors aux applications de communiquer avec n'importe quel dispositif en utilisant la description UPnP.
Notre approche se décompose en trois parties : la première consiste à générer automatiquement des ontologies à partir de l'annonciation des descriptions des dispositifs. Selon Kalfoglou[17], «Une on-

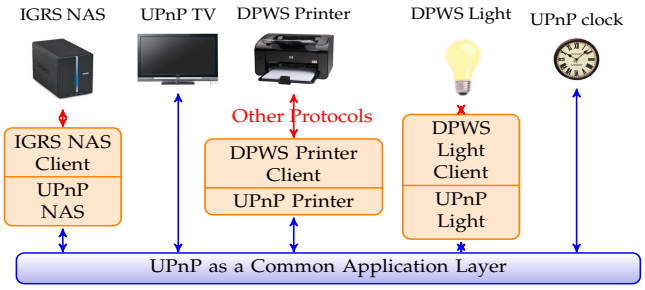

Figure 1: UPnP comme une couche applicative commune

tologie est une représentation d'une compréhension partagée des concepts importants dans un domaine d'intérêt». Dans notre contexte, le domaine d'intérêt est le réseau domestique et les concepts de l'ontologie sont les dispositifs, les services, les actions et les paramètres. Chaque dispositif réel est modélisé par une ontologie reflétant sa description spécifique.

Ensuite, nous appliquons des techniques d'alignements [11] d'ontologies afin de trouver des correspondances entre les descriptions de dispositifs équivalents par exemple entre une imprimante UPnP et une autre DPWS. L'alignement [11] est le processus de recherche d'un ensemble de correspondances entre deux entités provenant de deux ontologies, par exemple trouver des services équivalents, des actions et des paramètres entre deux descriptions des dispositifs. Une correspondance entre deux entités est représentée par une valeur de similarité normalisée dans un interval $\mathbb{R}^+[0,1]$.
La génération des ontologies est automatique alors que l'alignement d'ontologie est une étape semi-automatique qui nécessite l'intervention d'un expert humain chez l'opérateur ou le vendeur d'application afin de valider les alignements. Afin d'aider l'expert durant la validation, nous utilisons des techniques de détection de patron afin d'arranger automatiquement les actions qui sont totalement compatibles et celles dont l'expert devrait adapter en fonction des spécifications, valeurs par défaut, code d'adaptation, conversion de donnée etc. L'identification des patrons entre les correspondances des actions permet également d'accélérer la recherche dans l'ontologie lors de la génération de code.

Une fois les alignements validés, la dernière étape consiste à générer des mandataires automatiquement à partir des alignements qui contiennent les règles de transformation. Le mandataire généré s'annonce comme un équipement UPnP standard et translate les invocations vers le dispositif réel non-UPnP. Cette dernière étape repose sur l'Ingénierie Dirigée par les Modèles (IDM) qui est une méthode basée sur différents niveaux d'abstraction visant à accroître l'automatisation du développement logiciel. L'idée est de faire abstraction d'un domaine avec un modèle de haut niveau comme les ontologies, puis le transformer en un modèle de niveau inférieur jusqu'à ce que le modèle peut être rendu exécutable.

La suite de l'article s'organise comme suit, la section suivante présente une vue rapide sur les protocoles *plug and play* et l'état de l'art lié à l'adaptation des services. Notre approche d'adaptation est présentée dans la section 3. Nous présentons l'implémentation ainsi que son évaluation dans les sections 4 et 5. Nous terminerons par une discussion par rapport à l'état de l'art, une conclusion et les perspectives.

2. Etat de l'art

Les protocoles UPnP [26], IGRS [16] et DPWS [21] cohabitent dans les réseaux domestiques et partagent beaucoup de points communs. Ils sont tous axés sur les services avec des couches génériques au dessus de la couche IP ayant comme rôle : l'adressage, la découverte, la description, le contrôle et la notification. UPnP et IGRS utilise GENA (General Event Notification Architecture) pour la notification, SSDP (Simple Service Discovery Protocol) pour la découverte et SOAP pour l'invocation d'actions. DPWS se base sur un ensemble de web services standardisés (WS-*). Ils ciblent également des types d'appareils

similaires : UPnP et IGRS ciblent les appareils multimédia alors que le domaine de l'impression est dominé par UPnP et DPWS. Chaque protocole définit un profil standard pour les types d'appareil précisant une implémentation obligatoire et facultative pour les fabriquants.
Même si ces protocoles partagent beaucoup de fonctionnalités, les dispositifs ne peuvent pas coopérer en raison de deux différences principales : la description des services et les couches au-dessus d'IP. Pour résoudre l'hétérogénéité des couches au-dessus d'IP, le projet *Z2Z* [6] propose une approche de génération automatique des mandataires protocolaires, d'autres *framework* orienté service, reposant sur OSGi proposent une approche centralisée [15, 9] en utilisant des mandataire spécifiques pour cacher une telle diversité. Les mandataire des *framework* SOA représentent chaque appareil et sa description en tant que service local sur le *framework*. Un tel mandataire est appelé *Base Driver* [5], il répercute toute invocation locale sur le dispositif réel et vice-versa. Les *framework* orientés service couplés avec des *Base Driver* résolvent l'hétérogénéité des couches au-dessus d'IP mais celle de la description persiste.

Différentes approches ont été développées pour résoudre les questions liées au problème de l'interopérabilité, elles peuvent être classées dans les trois catégories suivantes :

- **Ontologie Commune** : *EASY* [4] et *MySIM* [15] ont travaillé sur la substitution des services. Les deux approches modélisent le domaine avec une ontologie commune qui détient les services et les propriétés qui les relient. Ainsi, la description de service doit être exposée en utilisant les mêmes entités de l'ontologie. Un service peut être remplaçable s'il est compatible avec un autre uniquement présent dans l'ontologie commune. Une modification du registre de service est nécessaire pour la recherche et/ou la publication d'un service car ces opérations dépendent à présent de l'ontologie commune. *MySim* effectue une adaptation dynamique soit par une redirection des appels ou une réplique de l'implémentation par une introspection du *byte code* des Jar/OSGi. Annoter la description manuellement est difficile et n'échappe pas aux erreurs. L'ajout d'un nouveau dispositif nécessite une mise à jour de l'ontologie commune qui se fait *manuellement* en ajoutant de nouveaux concepts et en les reliant à d'autres entités existantes dans l'ontologie. La mise à jour peut produire une ontologie incohérente car un nouveau type peut avoir une sémantique commune avec un ou plusieurs concepts existants [29].
  Dans notre approche, les ontologies sont générées **automatiquement** conforme à un méta-modèle à partir de la description du dispositif. Les correspondances entre les ontologies (UPnP et DPWS) générées automatiquement sont calculées semi-automatiquement par des techniques d'alignement.

- **Représentation Abstraite** : La deuxième catégorie modélise le domaine avec une représentation abstraite : une ontologie, comme dans *DOG* [9] et dans les travaux de *Coopman et. al*[8] de l'université de *Leuven* qui étendent l'ontologie commune SOUPA[7]. Dans *DOG* et dans les travaux de *Coopman et. al*[8], les appareils et actions similaires sont modélisés par une ontologie à travers des concepts abstraits, (dispositif de la lampe, interrupteur on/off) ainsi que leur messages d'interopérabilité et notifications (un dispositif de lampe est relié à un dispositif d'interruption). L'ontologie abstraite est interrogée pour générer des règles d'interopérabilité. Quand un message «off» est reçu par le *framework*, dans les deux travaux un module se charge de transformer les messages abstraits vers des invocations concrètes pour commander les dispositifs. Dans les travaux de *Coopman*, c'est l'*Orchestration Executer*, dans DOG ce sont les *Network Drivers* et le *Message Dispatcher*. L'interopérabilité entre les appareils est basée sur des messages de notifications abstraits et d'une association prédéfinie entre les commandes. Il y a un effort de développement afin de traduire les messages du haut niveau vers des invocations dépendantes du dispositif. *Coopman* fournit cette abstraction à l'utilisateur pour faciliter l'interaction avec les dispositifs. Dans les deux travaux l'approche est appliquée sur les appareils avec des invocations simples(On/Off) comme sur les lampes. *DogOnt*, l'ontologie commune détient beaucoup d'informations; spécialement des associations entre les actionneurs et les contrôleurs potentiels ce qui rend l'approche complexe pour des appareils simples. Cependant assurer l'interopérabilité des appareils complexes tels que des imprimantes avec des descriptions de 2000 lignes de code (LdC) n'est pas anodin surtout quand une action sur un dispositif est équivalente à une ou plusieurs actions sur un autre appareil. En outre, la correspondance entre le modèle abstrait et le modèle spécifique est effectuée manuellement par l'une des règles de transformation ou par des associations prédéfinies. Dans

notre approche, le modèle UPnP est choisi comme pivot pour sa large adoption dans le monde industriel, l'objectif principal de notre travail est la substitution de dispositifs disponibles DPWS par des dispositifs UPnP **standard** équivalents. Par rapport à ces deux approches, notre représentation commune est le pivot UPnP qui en l'alignant semi-automatiquement avec des descriptions de dispositifs équivalents, nous permet de générer automatiquement des mandataires à l'exécution pour résoudre l'hétérogénéité. En plus, la génération des ontologies est tout à fait automatique dans notre approche, ce qui n'est pas le cas dans les travaux utilisant une approche avec une ontologie abstraite dont la construction d'après [29] est une tâche difficile qui nécessite plusieurs itérations.

- **Langage Commun** : La troisième catégorie utilise un langage commun pour décrire des dispositifs avec la même sémantique, Moon et al. travaille sur l'*Universal Middleware Bridge* [20] (UMB), qui propose un *Unique Device Template* (UDT) pour décrire les dispositifs. Il maintient une table de correspondances qui contient un *mapping* entre la table locale et la table unique (UDT). UMB adopte un système centralisé, où chaque dispositif/réseau est enveloppé par un adaptateur qui convertit la table locale vers la table unique. Miori et al. [19] définit *DomoNet*, un *framework* d'interopérabilité qui se base sur les services Web et XML. Ils proposent *DomoML*, un langage standard pour décrire les dispositifs. Un *TechManager*, un par sous-réseau, traduit les capacités du dispositif en tant que Services Web, le mapping est effectué manuellement. L'approche *HomeSOA* [5] utilise les *Base Drivers* pour réifier les dispositifs comme services locaux, puis une autre couche de transformateurs *Refined Drivers* abstrait l'interface de service par type de dispositif unifié, par exemple deux lampes UPnP et DPWS sont représentées par la même interface *DimmingSwitch*. Ensuite, il appartient au développeur de tester le type de périphérique et d'invoquer l'action sous-jacente spécifique. Utiliser un langage commun dépend du *mapping* manuel entre les équipements du même type. Dans notre approche, nous utilisons une technique d'alignement d'ontologies pour trouver des correspondances semi-automatiquement entre les dispositifs équivalents.

3. Proposition

Cette section présente notre approche pour résoudre l'hétérogénéité entre les dispositifs. Nous commençons par les ontologies qui représentent les descriptions de chaque dispositifs. Ensuite par l'alignement d'ontologies qui permet de trouver les règles de correspondance pour passer d'une description à une autre. Enfin, nous détaillerons la détection de patrons qui assiste l'expert humain pendant la validation.

3.1. Besoin d'un méta modèle commun

Chaque protocole plug-n-play utilise son propre format d'annonciation pour diffuser les informations de son appareil ainsi que les services supportés : UPnP utilise un format XML propriétaire alors que DPWS et IGRS utilisent le langage standard WSDL (Web Service Description Langage). En plus, chaque protocole utilise une description avec une sémantique différente : UPnP utilise les mots clés `device`, `service`, `action` et `state variable`, alors que DPWS et IGRS (WSDL) utilisent `service`, `port`, `operation` et `message`. Afin de résoudre l'hétérogénéité de la description, nous avons appliqué des techniques du domaine de l'ingénierie des modèles. Nous définissons un méta modèle (couche M2) (figure 2, a) basé sur le modèle UPnP pour modéliser un appareil «plug-and-play». Nous utilisons les concepts comme suit : chaque appareil possède un ou plusieurs services, chaque service a une ou plus d'actions et chaque action dispose d'une variable d'état en entrée/sortie.

3.2. Génération automatique d'ontologie

Les appareils «plug-and-play» diffusent leur description sur le réseau. Des entités logicielles UPnP (respectivement DPWS) `Ontology Writers` sont abonnées aux services, par conséquent, à l'arrivée d'un service UPnP (respectivement DPWS) ces entités seront notifiées et pourront alors générer automatiquement des ontologies conformes au méta-modèle prédéfini à partir de la description annoncée par le dispositif. L'ontologie générée, représente alors le dispositif, les services, les actions et les variables.

Les `Ontology Writers` génèrent automatiquement une ontologie (couche M1) par type d'appareil qui est en conformité avec les concepts du méta modèle (voir figure 2, c, e). Ils génèrent également des ontologies à partir d'un fichier de description. Les `Ontology Writers` peuvent être placés sur

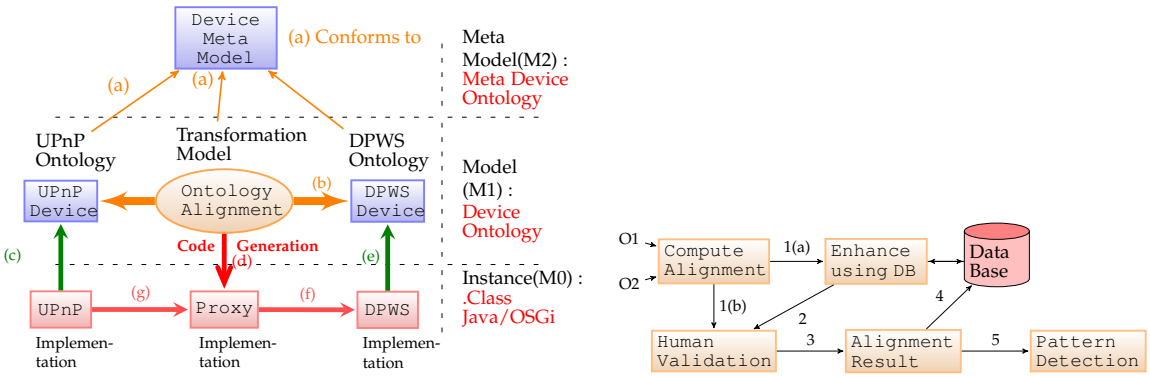

Figure 2: (a) Aperçu de l'approche (b) L'alignement

une *Set-Top-Box*(STB) et à chaque apparition d'un dispositif *plug and play* non identifié, ils génèrent des ontologies et les envoient à l'opérateur. Les `Writers` peuvent également être placés chez l'opérateur.

3.3. Alignement d'Ontologie

Maintenant que les modèles M1 sont générés et conformes à la couche M2, nous pouvons appliquer les techniques de transformation pour passer d'un modèle à l'autre. Dans l'ingénierie des modèles, des règles de transformation sont utilisées pour construire des ponts reliant les entités entre deux modèles existants. Dans notre approche, la traduction entre les dispositifs équivalents, services, actions et variables est assurée par des techniques semi-automatiques d'alignement d'ontologie (fig. 2 a, couche M1). Le paragraphe suivant résume l'alignement décrit dans nos travaux antérieurs [10] ainsi que les fonctionnalités ajoutées. L'*Aligner* est une combinaison des trois étapes suivantes :

Étape 1 : Le **Calcul de l'alignement** (figure 2, b) prend en entrée deux ontologies O_1 et O_2, puis applique les techniques d'alignement détaillés dans ([11], chapitre 4) comme *Leveinshtein*, *SMOA* etc. Ces techniques alignent deux actions comme un `SetLoadLevel` $\neq$ `GetLoadLevel`. Notre technique d'alignement SMOA++ qui est une amélioration de SMOA [24] utilise un dictionnaire sémantique *WordNet* [12] pour détecter les antonymes (`Set` $\neq$ `Get`) et les synonymes (`clock` $\equiv$ `timer`). Nous appliquons également une propagation de similarité entre les entités et améliorons par exemple deux similitudes des services si leur actions ont de fortes similitudes.

Étape 2 : La **validation** représente l'intervention humaine en utilisant une interface graphique pour valider ou modifier les correspondances. La sortie représente les règles de transformation pour générer automatiquement un mandataire qui résout l'hétérogénéité entre les dispositifs, elle est exprimée dans le langage d'ontologie standard OWL (*Web Ontology Language* [2]).

Étape 3 : L'**amélioration** des correspondances trouvées à l'aide d'une base de donnée. Cette étape est facultative et vise à améliorer les correspondances entre les entités ayant des valeurs en dessous d'un seuil prédéfini. Cette étape interroge la base de données à la recherche des correspondances préalablement validées et enregistrées. Après l'amélioration, l'alignement est re-validé. L'étape finale consiste à transformer les correspondances en axiomes d'ontologie. Finalement, des règles de détection de *patrons* sont appliquées pour détecter des correspondances d'actions complexes.

3.4. Détection de Patron

La détection de patron est une classification des données qui sont conformes à des propriétés et caractéristiques prédéfinies depuis un grand ensemble de donnée[14]. Nous utilisons les patrons pour classifier et répertorier les correspondances afin de détecter plus rapidement si les actions entre deux dispositifs sont compatibles et dans le cas contraire guider l'expert à les rendre valides (si possible).

Dans les cas simples, les actions ont des correspondances une-à-une, par exemple sur des lampes, `Switch(boolean true/false)` $\equiv$ `SetTarget(String On/Off)`. Toutefois, sur les imprimantes standards UPnP et DPWS [26, 18], l'action UPnP `CreateURIJob` est équivalente à l'association de deux actions DPWS `CreatePrintJob` et `SendDocument`. Le patron détecté en se basant sur les équiva-

lences de leurs paramètres est celui d'une union séquentielle, ceci implique qu'à l'invocation de l'action UPnP `CreateURIJob` sur le mandataire, ce dernier invoquera l'action DPWS `CreatePrintJob` en premier suivie de l'appel de l'action `SendDocument`.
Les patrons sont détectés automatiquement par des règles appliquées sur l'ontologie. Pour chaque patron identifié, l'ontologie est mise à jour par l'ajout de nouvelles propriétés entre les entités concernées.

3.4.1. Les patrons

Nous présentons dans cette sous-section les caractéristiques des patrons utile à notre approche. Les patrons sont détectés en utilisant des règles écrite dans le langage OPPL2 (Ontology Pre-Processor Language) [22], pour manque d'espace, nous présentons uniquement un exemple dans la figure 3, b.

Definition 1 (y = f(x))**.**

$$\forall x, y \in P/\ P\ ensemble\ de\ paramètres, f \in A/\ A\ ensemble\ d'actions$$

$$y = f(x) \iff f\ \textit{hasInput}\ x\ and\ \textit{hasOutput}\ y$$

Les patrons potentiels sont :

1. ***Direct_Mapping*** est un patron entre deux classes ayant une correspondance une-à-une. Cette propriété a les trois sous-propriétés suivantes :
   (a) ***Direct_Mapping_Input***: deux actions ayant au moins une paire équivalente de variables d'entrée, $f(x) \equiv g(x)$. Par exemple, sur les lampes, Switch (String ON/OFF) ≡ SetTarget(boolean true/false), voir figure 3, b qui montre la règle utilisée.
   (b) ***Direct_Mapping_Output***: deux actions ayant au moins une paire équivalente de variables de sortie. Les entrées sont négligées, `currentStatus=GetStatus()` ≡ `status=GetStatus()`.
   (c) ***Direct_Mapping_Input_Output***: est l'association des deux patrons précédents.
2. ***Union_Mapping***: une action équivalente à deux ou plusieurs actions sans un ordre prédéfini. Il y a trois cas pour un «union mapping».**One-to-N** :

   $$f(x, y)\ \ \textbf{\textit{Union\_to\_n}}\{g(x),\ h(y)\}$$

   Par exemple, `SetTime(hour, date)` est équivalente à l'union de `SetHour(hour)` et `SetDate(date)`. Les patrons N-to-One and N-to-M sont également détectés.
3. ***Sequential_Union***: est un «union mapping» ayant un ordre prédéfini entre les actions. Il y a trois cas pour une union séquentielle. **One-to-N** :

   $$(y = f(x))\ \ \textbf{\textit{Sequential\_to\_n}} \begin{cases} (1)\ \ a = g(x) \\ (2)\ \ y = h(a) \end{cases}$$

   Lorsqu'une application invoque $y = f(x)$, le mandataire invoque d'abord $a = g(x)$ puis $y = h(a)$ et retourne y. Ce patron existe entre les imprimantes UPnP et DPWS : CreateURIJob ≡ Sequential_Union (CreatePrintJob, SendDocument). N-to-One et N-to-M sont également détectés.

Les règles OPPL2 de détection de *patron* appliquées sur l'alignement mettent à jour les relations détectées en reliant les entités trouvées. Cette méta donnée est exploitée ensuite par les concepts de *matching* pour aider à la décidabilité des correspondances trouvées.

3.4.2. Concepts de Matching

Nous définissons *MConcept* pour classer les *mappings* dans l'un des quatre degrés du tableau 1. Ce concept s'applique sur deux séries d'actions S_a et S_b, des ontologies O_a et O_b qui sont reliées par des patrons *union*, *sequential* ou *simple mapping*.

- $\forall\, a \in A$, $np_{Input}(a)$ est le nombre de paramètres de l'action *a* en entrée.
  $np_{Input}(\sum_{i=1}^{n} a_i) = \sum_{i=1}^{n} np_{Input}(a_i)$.

- $\forall a, b \in A$, $nbMatched_{Input}(a, b)$ est le nombre de paramètres d'entrée équivalents entre *a* et *b*.
- $np_{Common}(a \cap b)$ est le nombre de paramètres communs entre les actions *a* et *b*.
- $Parameters_{Input} =$
  $np_{Input}(\sum_{i=1}^{n} a_i) + np_{Input}(\sum_{j=1}^{m} b_j) - np_{Common}(\cap_{i=1}^{n} a_i) - np_{Common}(\cap_{j=1}^{m} b_j)$.
- $nbMatched_{Input}(\sum_{i=1}^{n} a_i, \sum_{j=1}^{m} b_j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (nbMatched_{Input}(a_i, b_j))$.

Definition 2 (Matching, N-to-M mapping)**.**

$$\forall a_i, b_j \in A/\ n, m \in \mathbb{N}^{+*}, MConcept_{Input}(\sum_{i=1}^{n} a_i, \sum_{j=1}^{m} b_j) = \frac{nbMatched_{Input}(\sum_{i=1}^{n} a_i, \sum_{j=1}^{m} b_j) * 2}{Parameters_{Input}(\sum_{i=1}^{n} a_i, \sum_{j=1}^{m} b_j)}.$$

Table 1: Actions équivalentes pour les imprimantes Standards UPnP-DPWS

UPnP Action	DPWS Action	Matching	Décision
CreateURIJob	Sequential(CreatePrintJob, SendDocument)	$Sub._{In}$, $Ex._{Out}$	?
CancelJob	CancelJob	$Ex._{In}$, $Ex._{Out}$	✓
GetPrinterAttributes	Union(GetPrinterElements, GetActiveJobs)	$Sub._{In}$, $Sub._{Out}$	?
GetJobAttributes	GetJobElements	$Ex._{In}$, $Sub._{Out}$	✓

Une fois le *MConcept* calculé, on classe les correspondances comme suit :

- $ExactMatch_{Input}(S_a, S_b)$: pour **tout** paramètre d'entrée de S_a il y a une relation *equivalentTo* avec **tout** paramètre d'entrée de S_b. Par exemple, f(x,y) et g(x,y). Si $MConcept_{Input}(S_a, S_b) = 1$.
- $PlugIn_{Input}(S_a, S_b)$: pour **certains** paramètres de S_a il y a une relation *equivalentTo* avec **chaque** paramètres d'entrées de S_b. Par exemple `f(x,y,z) ≡ g(x,y)`. Le *PlugIn* est satisfait ssi : $\{(MConcept_{Input}(S_a, S_b) \neq 1) \wedge (np_{Input}(S_b) = nbMatched_{Input}(S_a, S_b)) \wedge (np_{Input}(S_a) > np_{Input}(S_b))\}$. Le paramètre d'entrée `z` de l'action `f` peut être ignoré.
- $Subsume_{Input}(S_a, S_b)$: pour **chaque** paramètre d'entrée de S_a il y a une relation *equivalentTo* avec **certains** paramètres d'entrée de S_b. Par exemple, `f(x,y) ≡ g(x,y,z)`. Le *Subsume* est satisfait ssi : $\{(MConcept_{Input}(S_a, S_b) \neq 1) \wedge (np_{Input}(S_a) = nbMatched_{Input}(S_a, S_b)) \wedge (np_{Input}(S_a) < np_{Input}(S_b))\}$. Le paramètre `z` ne peut pas être ignoré. L'expert qui valide les alignements vérifie les spécifications pour savoir si `z` peut prendre une valeur par défaut. Sur une imprimante DPWS, l'action *SendDocument* a une variable d'entrée *LastDocument* qui peut avoir la valeur *true* par défaut, ceci ne sera pas en conflit avec le comportement d'une imprimante UPnP.
- $Unknown_{Input}(S_a, S_b)$: pour **certains** paramètres d'entrée de S_a il y a une relation *equivalentTo* avec certains paramètres de S_b et qui ne vérifie aucun degré de match précedent. Par exemple, f(x,y,z) et g(x,b,c). *Unknown* est satisfait ssi : $(MConcept_{Input}(S_a, S_b) \neq 1) \wedge$
  $\neg(ExactMatch_{Input}(S_a, S_b) \wedge PlugIn_{Input}(S_a, S_b) \wedge Subsume_{Input}(S_a, S_b))$.

Le tableau de la figure (3, a) est utilisé pour décider si le mapping entre les actions est une réussite ou un échec. Le tableau 1, montre le résultat des imprimantes, l'expert se concentre alors sur les *undefined* trouvées, il vérifie selon les spécifications du dispositif si certains paramètres peuvent avoir des entrées ou sorties par défaut. Ensuite, l'expert décide si c'est un succès ou un échec. Les patrons permettent également d'accélérer la génération de code, par exemple dans le cas d'un direct input pas besoin d'interroger l'alignement pour savoir s'il y a un code à générer pour les retours des invocations.
L'étape d'alignement et de validation est effectuée chez l'opérateur, les alignements validés sont utilisés pour la génération de code afin de passer d'une représentation des dispositifs avec un langage de haut niveau indépendant d'une technologie vers des mandataires qui s'exécutent et interagissent avec des dispositifs réels.

(S_a, S_b)	$Exact_{In}$	$PlugIn_{In}$	$Subsume_{In}$	$Unknown_{In}$
$Exact_{Out}$	✓	✓	?	?
$PlugIn_{Out}$	x	x	x	x
$Subsume_{Out}$	✓	✓	?	?
$Unknown_{Out}$	x	x	x	x

```
?f:CLASS, ?g:CLASS, ?x1:CLASS, ?x2:CLASS
SELECT ?f subClassOf hasUPnPInput some ?x1
?g subClassOf hasDPWSInput some ?x2,
?f equivalentTo ?g,
?x1 equivalentTo ?x2
BEGIN
ADD ?f subClassOf SimpleMappInput some ?g
END;
```

Figure 3: (a) Table de décision, (✓ :success, x :fail, ? :undefined) (b) Détection de pattern, Règle OPPL2

4. Implémentation

Pour l'implémentation, les *base drivers* d'UPnP Felix Apache [3] et DPWS SOA4D [23] ont été utilisés. Nous avons développé les *Ontology Writers* sur Felix/OSGi en utilisant l'API OWL 3.0 [2] et mis en œuvre notre algorithme d'alignement en utilisant l'API d'alignement 4.0 [1] et WordNet. Nous avons également développé une interface graphique à l'aide de SWING au dessus du serveur d'alignement pour valider, ajouter, supprimer, modifier, enregistrer les correspondances dans une base de données (MySQL). L'interface graphique permet à l'expert d'ajouter des méta données aux alignements comme les valeurs par défaut. L'expert sélectionne simplement deux entités et ajoute un trait pour ajouter une propriété d'équivalence. Les méthodes d'alignement supportées à présent sont SMOA et SMOA++. Les règles en OPPL2 [22] ont été utilisées pour détecter les patrons.

Pour la génération automatique des mandataires, nous avons implémenté DOXEN «a Dynamic Ontology-based proXy gENerator» qui est un générateur automatique du mandataire à partir d'un alignement d'ontologie qui représente le langage de transformation. Un mandataire est généré (figure 2 a, couche M0) pour chaque type d'appareil non-UPnP ayant une équivalence avec un appareil standard UPnP. Le mandataire publie un profil UPnP standard ce qui implique que les applications déjà développées et qui ciblent les appareils UPnP standard interagissent sans aucune modification avec les appareils non-UPnP à travers le mandataire. Ce dernier redirige les invocations vers l'appareil non-UPnP.

DOXEN génère automatiquement les mandataires en fonction des appareils non-UPnP présents sur le

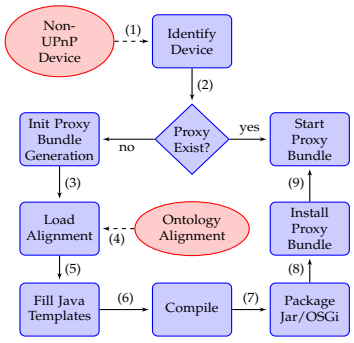

Figure 4: Diagramme de Générateur de Mandataire

réseau. DOXEN est un bundle OSGi qui est installé et lancé par l'opérateur dans la maison sur une STB par exemple. A son démarrage, DOXEN analyse un fichier de configuration contenant des informations sur les dispositifs équivalents et les fichiers d'alignements. Le générateur (figure 4, a) construit un mandataire à la découverte d'un dispositif non-UPnP ayant un profil équivalent UPnP. Il est notifié dès l'arrivée des services non-UPnP (étape (1)). Il vérifie (2) le modèle d'appareil et le numéro de version. En fonction du type du périphérique non-UPnP, DOXEN charge le fichier d'alignement d'ontologie correspondant (4), le parcourt et remplit (5) les *templates* FreeMarker [13] Java pré-écrites pour produire

des fichiers Java. Une fois les fichiers générés, DOXEN les compile (6) à l'exécution avec Janino [25] et génère un *bundle* OSGi/Jar (7). Ensuite, il installe (8) et démarre (9) le module généré qui représente le mandataire UPnP qui dès son lancement interroge l'appareil non-UPnP et récupère les informations générales (fabricant, version, etc) et les diffuse lors de l'annonciation de sa description sur le réseau.
Dès que l'appareil non-UPnP quitte le réseau, le mandataire bundle OSGi change son état de `Started` à `Installed` par conséquent le mandataire UPnP disparait de la plateforme OSGi. A l'apparition d'un dispositif non-UPnP équivalent et dont l'alignement existe, si un mandataire est déjà généré par DOXEN, alors il suffit de le démarrer. Sinon DOXEN génère le mandataire.

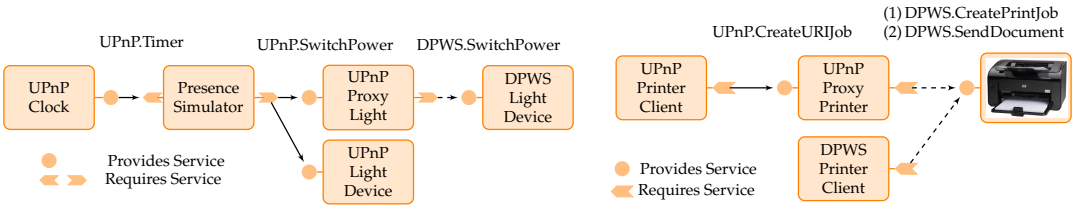

Figure 5: (a) Application Presence Simulator (b) Evaluation sur une imprimante DPWS

Nous avons implémenté une application de «Presence Simulator» sur OSGi (figure 5, a) qui détecte une horloge UPnP et active/désactive toutes les lumières à un moment donné de la journée. Nous avons utilisé une lampe UPnP Felix et une lampe SOA4D DPWS [23]. L'application contrôle les lampes DPWS comme des lampes UPnP à travers le mandataire UPnP généré automatiquement. Le processus d'adaptation est transparent pour l'application, le mandataire est généré en utilisant le fichier d'alignement et installé à l'exécution. Le framework OSGi, les dispositifs et les applications sont déployés sur un PC tandis que DOXEN est déployé sur une *Set-Top-Box*. Nous avons testé notre approche sur une imprimante **HP 4515x** qui implémente le profil standard [18] DPWS. Le mandataire généré automatiquement est publié en tant qu'imprimante UPnP avec un profil d'impression standard [27], voir tableau 1. Nous sommes en mesure d'imprimer un fichier (pdf, txt, ps), d'annuler et consulter une tâche à travers un *Control Point* UPnP [28].

5. Evaluation

Nous avons testé notre approche sur un PC avec un processeur Intel x86 Centrino Core Duo, 2 GHz de fréquence d'horloge et une capacité de 1 Go de RAM. Les «Ontology Writers» ont généré des ontologies pour les lampes, horloges et imprimantes. Le tableau de la figure(6,b) révèle le temps en secondes de la génération d'ontologie, les lignes de code (LdC) de l'ontologie ainsi que les LdC de la description des dispositifs. La différence du temps de construction des ontologies entre UPnP et DPWS est due au suivant : d'une part les équipements DPWS ont des paramètres complexes hiérarchique exprimés en WSDL et XSD (Voir la description WSDL d'une imprimante [18]), tandis que la description UPnP est plus simple avec des paramètres sans structure hiérarchique. D'autre part, les «base drivers» [3, 5] représentent comme services locaux OSGi les équipements UPnP ou DPWS sur le réseau. Les développeurs du SOA4D DPWS Base Driver [23] ont choisi de récupérer à partir d'un équipement DPWS uniquement les informations de l'appareil et les services en ignorant les opérations et les noms et types des paramètres. Leur choix est motivé par le fait que les applications qui invoquent une opération sur un dispositif DPWS connait déjà le nom et les paramètres. Pour générer une ontologie à partir d'un équipement DPWS il faut récupérer le fichier WSDL hébergé qui peut être d'une taille et complexité non négligeable. Pour UPnP toutes les informations sont déjà présentes localement sur le «base driver» ce qui permet une génération plus rapide. La différence de temps peut être réduite en améliorant le base driver DPWS et en représentant la description entière de l'équipement localement. Nous avons testé l'alignement sur les différents dispositifs en appliquant l'algorithme SMOA++ [10] et SMOA [24]. Le

tableau de la figure(6,a) résume l'évaluation de l'alignement sur trois dispositifs en utilisant SMOA [24] et SMOA++ [10]. Il expose la méthode utilisée, le temps de l'alignement en secondes, le pourcentage du succès pour un seuil S. (s1 = 0,65) et S. (s2 = 0,2) ainsi que le nombre de fausses f. correspondances (s1) et f. (s2). La méthode SMOA++ utilise un dictionnaire sémantique externe WordNet et montre de meilleurs résultats sur des ontologies de relativement petite taille (horloges et lampes). Sur des ontologies plus complexes (imprimante) les résultats sont légèrement meilleurs avec un seuil de 0,2 et le nombre de fausses correspondances est un peu inférieur avec un seuil de 0,6. Il est évident que SMOA++ passe plus de temps que la méthode SMOA, ceci est du à l'accès et la recherche dans le dictionnaire sémantique WordNet et à l'amélioration structurelle. La différence reste acceptable car l'alignement est effectué hors ligne du côté de l'opérateur.

Type	méthode	temps (sec.)	S.(s1=0,65)	f.(s1)	S.(s2=0,2)	f.(s2)
Printer	smoa++	421	81%	3	**85%**	**12**
	smoa	71	61%	6	81%	11
Light	smoa++	9	63%	2	**72%**	3
	smoa	1,5	54%	4	**72%**	4
Clock	smoa++	2	66%	1	**83%**	1
	smoa	0,4	66%	1	66%	1

Figure 6: (a) Evaluations des Alignements sur PC

Type	temps (sec)	Desc.(LdC)	OWL (LdC)
Printer UPnP	0.5	610	1573
Printer DPWS	187	2237	9082
Light UPnP	0.2	51	365
Light DPWS	0.8	213	245
Clock UPnP	0.05	49	161
Clock DPWS	0.25	48	123

(b) Ontologies Générées sur PC

Le tableau (7, a) reprend le temps mis par DOXEN déployé sur un Felix/OSGi qui se trouve sur une STB *SodaVille* Intel Atom 1.2GHz avec 384 MB de RAM pour générer du code, le compiler, l'assembler dans les Jar et enfin l'installer. Nous montrons aussi le nombre de fichiers *templates* de Java et les lignes de code (LdC) utilisées ainsi que le nombre de fichiers Java générés et leur ligne de code. Le tableau (fig. 7, b) indique le temps d'invocation d'un client DPWS invoquant directement les actions sur l'imprimante HP 4515x et d'un client UPnP l'invoquant via un mandataire généré UPnP comme le montre la figure(5, b). Les clients (UPnP, DPWS) et le mandataire généré sont déployés sur la même STB et connectés à l'imprimante via le réseau *wifi*. Les clients impriment le même fichier. Les résultats montrent une petite différence qui représente le temps de réorienter l'invocation d'action et de remplissage des paramètres avec les valeurs appropriées.

Proxy	Temps (sec)	Fichiers	LdC
Templates	–	7	821
Printer	25	30	3054
Light	5.4	14	1569
Clock	2.8	8	912

Figure 7: (a) Proxy Générés sur STB

Actions DPWS	Client DPWS	Client UPnP
(CreatePrintJob, SendDocument)	844	1141
CancelJob	40	57
(GetPrinterElements, GetActiveJobs)	357	583
GetJobElements	364	481

(b) Temps d'invocation en ms sur une DPWS HP 4515x

6. Discussion

Dans les travaux connexes, l'approche intégrée avec une ontologie commune est appliquée à des dispositifs relativement simples (lampes, interrupteurs et services simples) par conséquent les ontologies proposées dans leur approche sont relativement simples et petites comparées à des dispositifs plus complexes comme des imprimantes ayant une description avec 2237 lignes de code (LdC) pour une imprimante DPWS [18]). Les ontologies générées dans notre approche pour les imprimantes sont relativement complexes. Par conséquent, la construction manuelle de grandes ontologies nécessite un effort non négligeable [29]. Dans notre approche, les équipements *plug-and-play* annoncent leur description,

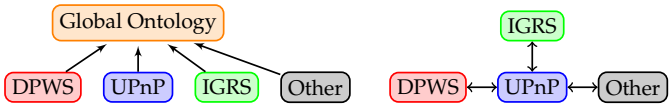

Figure 8: (a) Commune (b) Federée (Alignement)

alors les «Ontology Writers» UPnP et DPWS génèrent automatiquement des ontologies conforme à un méta modèle prédéfini. Le temps nécessaire pour la génération de l'ontologie est de l'ordre de 3 minutes (187 sec) pour une imprimante DPWS alors que l'alignement des imprimantes nécessitent 421 sec ($\approx$ 7 minutes) d'après les tableaux de la figure 6, ceci reste acceptable et plus rapide que de construire les ontologies manuellement, comme suggéré dans l'état de l'art. À notre connaissance, nous sommes les premiers à résoudre l'hétérogénéité entre deux imprimantes standard en utilisant des techniques d'alignement d'ontologies. Utiliser une ontologie commune pour les équipements équivalents semble être trop optimiste parmi les concurrents tels que les fabricants et les comités de normalisation, il n'y a jamais eu une description unifiée ou un profil commun proposé pour le même type d'appareil (UPnP, DPWS ou IGRS). Imposer l'utilisation d'une ontologie commune revient à imposer l'utilisation d'un profil standard unifié.
Les spécifications sont indépendantes du protocole et de la technologie donc l'expert qui effectue la validation hors ligne du côté de l'opérateur à l'aide de la GUI peut être un technicien ou un expert du domaine. Pour la validation d'imprimante nous avons validé l'alignement en nous référant aux profils standards des imprimantes UPnP et DPWS [18, 27]. L'alignement validé peut ensuite être déployé sur la passerelle du client afin qu'il puisse être utilisé plus tard par DOXEN qui est déployé sur une *set-top box*. Les principaux avantages de cette approche peuvent se résumer comme suit : **premièrement**, les applications déjà développées, qui ne visent que les appareils standards UPnP peuvent interagir avec d'autres appareils non-UPnP standards grâce au mandataire généré dynamiquement. En plus, les développeurs des applications peuvent se concentrer sur un seul protocole et ne pas prendre en compte les autres protocoles et appareils non-UPnP. **Deuxièmement**, il n'est pas nécessaire d'ajouter des piles réseau supplémentaires à l'appui d'autres protocoles sur des dispositifs qui hébergent les applications de contrôle. La pile UPnP déjà déployée est utilisée pour interagir avec les autres appareils non-UPnP via le mandataire UPnP. **En plus**, le processus de construction des ontologies est relativement simple et plus rapide que la construction manuelle d'une ontologie commune globale (comme dans DOG, MySim, EASY, etc) spécialement lorsqu'il s'agit de dispositifs complexes, comme les imprimantes. L'expert qui valide l'alignement à l'aide de notre interface graphique se contente juste d'enlever ou d'ajouter des traits pour relier deux entités équivalentes. Il peut également ajouter des valeurs par défaut. **Enfin**, les ontologies sont indépendantes et peuvent être réutilisées si un autre protocole (que UPnP) est choisi comme pivot. Dans ce cas, il suffit d'aligner les ontologies avec celles du pivot. Par rapport aux autres approches, notre proposition fournit un ensemble de module (génération d'ontologie et mandataire) où l'expert n'intervient que pour valider des correspondances semi-automatique.

7. Conclusion et travaux futurs

Dans cet article nous proposons une approche basée sur la génération de code à partir des alignements d'ontologies pour résoudre l'hétérogénéité des services et protocoles. Tout d'abord, nous générons automatiquement pour chaque type d'appareil une ontologie conforme à un méta-modèle, puis nous appliquons des techniques semi-automatiques d'alignement d'ontologies pour trouver les correspondances entre les services équivalents. Suite à la validation de l'expert et l'édition avec une interface graphique, nous appliquons des techniques de reconnaissance de patrons pour classer les actions équivalentes et les compositions. L'alignement qui en résulte représente un ensemble de règles de transformation utilisé pour générer à l'exécution un mandataire spécifique à partir de *templates* existants. La génération de mandataire est déclenchée à l'arrivée des équipements ou à la demande. Le mandataire généré est exposé comme un équipement **standard** UPnP qui transfère les invocations aux équipements non-UPnP concernés. Nous avons choisi d'exposer les périphériques non-UPnP comme

étant des équipements UPnP, le protocole le plus mûre à ce jour. Nous avons testé l'approche sur une imprimante DPWS **HP 4515x**. La solution devrait fonctionner sur les dispositifs IGRS car ils utilisent les mêmes couches protocolaires que UPnP et exposent les services avec un WSDL. Actuellement, nous travaillons l'ajout d'autres opérations lors de la validation de l'alignement à l'aide de la GUI, par exemple, la transformation d'unité entre deux capteurs de température $°C = (5/9)°F$.

Bibliographie

1. Alignment api. http://alignapi.gforge.inria.fr. [acc. 1-Dec-2010].
2. The owl api. http://owlapi.sourceforge.net/. [acc. 1-Dec-2010].
3. Felix Apache. Base driver. http://felix.apache.org/site/apache-felix-upnp.html. [acc. 1-Dec-2010].
4. S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Efficient semantic service discovery in pervasive computing environments. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, 2006.
5. A. Bottaro and A. Gérodolle. Home soa facing protocol heterogeneity in pervasive applications. In *Proceedings of the 5th international conference on Pervasive services*, 2008.
6. Y. Bromberg, L. Réveillère, J. Lawall, and G. Muller. Automatic generation of network protocol gateways. Middleware '09. Springer-Verlag New York, Inc.
7. H. Chen, T. Finin, and A. Joshi. *The SOUPA Ontology for Pervasive Computing*. Springer, 2005.
8. T. Coopman, W. Theetaert, D. Preuveneers, and Y. Berbers. A user-oriented and context-aware service orchestration framework for dynamic home automation systems. In *Ambient Intelligence and Future Trends - International Symposium on Ambient Intelligence*. Springer, 2010.
9. Domestic-OSGi-Gateway. http://elite.polito.it/dog-tools-72.
10. C. El Kaed, Y. Denneulin, FG Ottogalli, and LF Melo Mora. Combining ontology alignment with model driven engineering techniques for home devices interoperability. *SEUS*, 2010.
11. J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, 2007.
12. C. Fellbaum. Wordnet: An electronic lexical database. Cambridge, MA: MIT Press, 1998.
13. FreeMarker. Template engine manual. http://freemarker.org, 2009.
14. M.G Gonzalez, R.C.Thomas. *Syntatic Pattern Recognition:an Introduction*. Addison Wesley,Reading,MA, 1978.
15. N. Ibrahim, F. Le Mouël, and S. Frénot. User-excentric service composition in pervasive environments. *AINA*, 2010.
16. IGRS. http://www.igrs.org/. [acc. 1-Dec-2010].
17. Y. Kalfoglou. Exploring ontologies. *in Handbook of Software and Knowledge Engineering*, 2001.
18. Microsoft. Standard dpws printer specifications, January 2007.
19. V. Miori, L. Tarrini, M. Manca, and G. Tolomei. An open standard solution for domotic interoperability. *Transactions on Consumer Electronics*, 2006.
20. KD Moon, YH Lee, and CE Lee. Design of a universal middleware bridge for device interoperability in heterogeneous home network middleware. In *Transactions on Consumer Electronics*, 2005.
21. OASIS. Devices profile for web services version 1.1, 2009.
22. OPPL2. The ontology pre-processing language 2. http://oppl2.sourceforge.net/.
23. SOA4D. https://forge.soa4d.org/. [acc. 1-Dec-2010].
24. G. Stoilos. A string metric for ontology alignment. *International Semantic Web Conference*, 2005.
25. Arno Unkrig. Janino compiler. www.janino.net.
26. UPnP. http://www.upnp.org/. [acc. 1-Dec-2010].
27. UPnP. Standard upnp printer, October 28 2006.
28. UPnP-Felix-Apache. http://felix.apache.org/site/apache-felix-upnp.html.
29. H. Wache, U. Visser, and T. Scholz. Ontology construction - an iterative and dynamic task. *Florida Artificial Intelligence Research Society Conference (FLAIRS), Pensacola, FL, USA.*, 2002.
30. M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun.*, 1999.