Modéliser et spécifier des services domotiques avec une approche objet

Lydie du Bousquet* – Ben Yan – Masahide Nakamura** – Ken-ichi Matsumoto****

** Universités de Grenoble (UJF - CNRS)*
Laboratoire d'Informatique de Grenoble (LIG)
BP. 72, 38402 Saint-Martin d'Hères cedex, FRANCE

*** Nara Institute of Science and Technology (NAIST)*
8916-5, Takayama-cho, Ikoma-shi, Nara, 630-0192 Japan

lydie.du-bousquet@imag.fr; {hon-e, masa-n, matumoto}@is.naist.jp

RÉSUMÉ. *Assurer la correction de services domotiques est un enjeu fondamental pour garantir la qualité de vie dans une « maison intelligente ». Dans cet article, nous proposons un cadre pour modéliser et spécifier des services domotiques. Le modèle s'appuie sur une conception objet et est implanté en Java. Les propriétés attendues des services sont elles exprimées sous la forme d'assertions en JML (Java Modeling Language). Ces assertions peuvent être utilisées comme oracle pendant les phases de test. Les propriétés représentent trois niveaux de spécification. Les propriétés locales visent à spécifier qu'un appareil est utilisé selon le manuel d'utilisation. Les propriétés globales spécifient qu'un service domotique agit comme attendu. Les propriétés « issues de l'environnement » permettent de décrire les contraintes imposées par les règles de vie dans la maison.*

ABSTRACT. *Assuring correctness is a crucial issue to guarantee high quality of life in smart home. In this paper, we propose a modeling/validation framework for the home network services. We first introduce an object-oriented modeling technique to clarify the relationships among the appliances, the services and the home (environment) objects. We then express expected properties with Java Modeling Language (JML). Those properties are represented as JML assertions, and are embedded within Java source code of the appliance, the service and the home objects, respectively. These assertions can be used as oracle for testing activities. Three kinds of correctness have to be satisfied by networked system. The* local correctness *is defined by safety instructions of individual networked appliances. The* global correctness *is specified as required properties of services. The* environment correctness *is prescribed as residential constraints in home and surrounding environments.*

MOTS-CLÉS : *Services domotiques, modélisation objet, JML*

KEYWORDS: *Home network services, object-oriented modelisation, JML*

17e soumission à *MSR 2007*, le 26 février 2007.

1. Introduction

La domotique regroupe l'ensemble des techniques et technologies permettant de superviser, d'automatiser, de programmer et de coordonner les tâches de confort, de sécurité, de maintenance et plus généralement de services dans l'habitat. Les applications les plus courantes sont : l'optimisation d'énergie, la sécurité des biens et des personnes, l'ambiance lumineuse, la régulation d'électricité, de chauffage ou de climatisation, la simulation de présence, etc. Ces applications pilotent les différents appareils communicants de la maison. Par exemple, un service « Home cinéma » coordonne le fonctionnement de la télévision, du lecteur de DVD, des enceintes, des lumières et des rideaux, afin d'offrir une ambiance « salle de cinéma » à la suite d'une simple requête. Un système domotique est réactif et temps-réel par définition.

Pour assurer le développement et la commercialisation de tels services, il faut démontrer que ces services sont à la fois corrects (ils ont exactement ce pourquoi ils ont été définis) et qu'ils ne sont pas « dangereux » (ils mettent pas la vie des utilisateurs en danger par exemple). On peut exprimer ces propriétés à trois niveaux. Tout d'abord, chaque appareil doit être utilisé selon les recommandations qui lui sont propres. Elles sont en générale décrites dans la notice d'utilisation. De plus, sachant qu'un service coordonne souvent différents appareils, il faut veiller à ce que leur exécution en parallèle ne conduise pas à des conflits d'utilisation. La mise en route simultanée d'une machine à laver et d'un four pourrait provoquer une coupure électrique au niveau du disjoncteur. D'autres situations plus graves pourraient êtres provoquées comme une absence de déverrouillage des portes et fenêtres en cas d'incendie.

A notre connaissance, il existe pas ou peu d'études sur la validation de systèmes domotiques. Dans cet article, nous exprimons les propriétés attendues à trois niveaux. On appelle « propriétés locales » les instructions relatives à la bonne utilisation d'un appareil en particulier. On appelle « propriétés globales » les propriétés attendues au niveau d'un service impliquant différents appareils. Enfin, on appelle « propriétés issues de l'environnement » les propriétés et contraintes imposées au niveau de la maison et son environnement dans leur ensemble.

Dans cet article, nous proposons un cadre pour modéliser et valider des services. On utilise une démarche objet pour modéliser les appareils, les services et la maison. Les propriétés locales, globales et issues de l'environnement sont exprimées sous la forme d'assertions (invariants, pré et post-conditions) respectivement au niveau des appareils, des services et de la maison. D'un point de vue pratique, le modèle est écrit en Java et les propriétés en JML (Java Modeling Language). Le choix de Java et JML ont été guidé par l'expérience respective des auteurs, d'une part sur l'utilisation de Java pour implanter des services domotiques grandeur réelle [NAK 05, LEE 05, NAK 06],

et d'autre part sur l'utilisation de JML pour tester et vérifier des applications Java [BOU 04, LED 04, BOU 07b].

Dans la suite, nous présentons brièvement quelques exemples de services domotiques. Puis, dans le paragraphe 3, nous exprimons les propriétés attendus pour un système domotique. Le paragraphe 4 presente un modèle objet. Dans le paragraphe 5, on décrit les principes de base de JML et l'expression des propriétés du modèle dans ce langage. Le paragraphe 6 décrit la démarche appliquée pour la validation par le test. Le paragraphe 7 aborde les travaux connexes. Enfin, le paragraphe 8 conclut et dresse quelques perspectives.

2. Préliminaires

2.1. *Un système domotique*

Dans le cadre de cet article, un système domotique consiste en un ou plusieurs appareil(s) connectés via un réseau local (LAN) au sein de la maison. Chaque appareil propose une interface (*application program interface : API*), par laquelle il peut être contrôlé par l'utilisateur ou un programme externe. L'ensemble des appareils connectés est contrôlé par un serveur. Les services et applications sont installés sur le serveur. Un service coordonne le fonctionnement des différents appareils. Un tel service est implanté sous la forme d'une application logicielle utilisant les APIs des appareils.

2.2. *Exemples de services domotiques*

Nous introduisons ici quatre exemples de services.

S1 **le service Home-Cinéma**. Ce service contrôle et coordonne l'exécution de la télévision, du lecteur de DVD, les enceintes associées, de lampes et d'un ensemble de rideaux. Le but de ce service est de créer une ambiance salle de cinéma dans la pièce considérée en « une seule requête ». À la demande de l'utilisateur, le service allume la télévision sur l'entrée DVD, ferme les volets, allume le système d'enceintes, éteint les lumières et allume le DVD en lecture.

S2 **le service de relaxation**. Ce service contrôle et coordonne l'exécution du lecteur de DVD, des enceintes, des lumières, du climatiseur et de la bouilloire électrique. Le but de ce service est de fournir à l'utilisateur une ambiance relaxante dans le salon. À la demande de l'utilisateur, le service allume le lecteur de DVD et les enceintes pour proposer ambiance musicale douce, le niveau de lumière est ajusté, le climatiseur est réglé pour fournir une température confortable et la bouilloire électrique est allumée pour la préparation d'eau chaude pour le thé.

S3 **le service douche**. Ce service contrôle et coordonne l'exécution du chauffe-eau à gaz, de l'ouverture du robinet de douche et du chauffage (ou

```
Public   DVDTheaterService {
         DigitalTV     tv = new DigitalTV();
         DVDPlayer  dvd = new DVDPlayer();
         SoundSystem  sound =new SoundSystem();
         Light   light = new Light();
         Curtain  curtain = new   Curtain();

         tv.on();                          /* Turn on TV */
         tv.setVisualInput('DVD');

         dvd.on();                         /* Turn on the DVD Player */
         dvd.setSoundOutput('5.1');

         sound.on();                       /* Turn on the Sound System */
         sound.setInputSource('DVD');
         sound.setVolumeLevel(25);

         curtain.closeCurtain();           /*  Close curtain  */

         light.setBrightnessLevel(1);      /*  Minimize brightness  */

         tv.playTv();                      /*  Play TV   */
         dvd.playDvd();                    /*  Play DVD  */
}
```

Figure 1. *Pseudo-code pour le service Home-Cinéma*

climatiseur). Le but de ce service est de créer une ambiance confortable dans la salle de bain. À la demande de l'utilisateur, le service allume le chauffe-eau pour préchauffer l'eau. Lorsque l'utilisateur entre dans la salle de bain, le service commute le robinet de la baignoire en position douche et ouvre le robinet. Le chauffage (ou climatiseur) est réglé pour obtenir une température agréable dans la salle de bain.

S4 **le service de préparation de la cuisine**. Ce service contrôle et coordonne l'exécution de l'ouverture du robinet d'arrivée de gaz, l'aération, le four, et les lumières de la cuisine. Le but de ce service est d'organiser la cuisine en vue de la préparation d'un repas. À la demande de l'utilisateur, la lumière de la cuisine est allumée, le robinet d'arrivée de gaz est ouvert, le ventilateur est allumé et le four est mis en position préchauffage.

Par soucis de simplification, les quatre services ci-dessus sont présentés sous forme élémentaire. En ajoutant différents capteurs, il est possible de rendre ces services plus adaptés aux besoins et/ou plus soucieux de l'optimisation d'énergie. Par exemple, pour le service S1 (Home-Cinéma), on peut imaginer que le son est réglé en fonction de l'heure. De même, le réglage du chauffage peut être conditionné par la température ambiante de la pièce ou par la saison, et celui des lampes peut-être conditionné par la luminosité ambiante (mesurable à l'aide d'un luxmètre).

Fig. 1 illustre le principe du code du service S1 home-cinéma (*DVDTheaterService*). Le service en lui-même consiste en une suite d'appels aux méthodes des différents appareils (via leurs API).

3. Expression des propriétés attendues pour les services domotiques

Naturellement, il est attendu qu'un service ou plus généralement qu'un système domotique fasse correctement ce pourquoi il a été conçu (correction fonctionnelle). Il est assez facile d'exprimer intuitivement les propriétés correspondantes. Pour le service S1 Home-Cinéma, pendant la lecture du film, le lecteur de DVD, la télévision, le système d'enceintes doivent rester allumés, les lumières et les rideaux doivent rester fermés (sauf événement le justifiant).

Par ailleurs, il existe un certain nombre de propriétés attendues d'un système domotique en général. En effet, un système domotique ne doit pas causer (1) de blessures aux habitants de la maison (ou pire leur mort) , de leurs voisins ou toute autre forme de vie, et/ou (2) de dommages matériels. Il en va de la « sûreté » de l'utilisateur. De plus, il est possible de décrire des contraintes d'usage sur une habitation (limitation du bruit au delà d'une certaine heure). Etant souvent implicites, il est difficile de cerner l'ensemble de ces propriétés.

Une partie du travail d'analyse consiste à extraire puis à exprimer les propriétés (fonctionnelles ou de sûreté). On se place pour cela selon trois points de vue : les appareils, les services et la maison. Ceci nous permet de distinguer respectivement les propriétés locales, globales et issues de l'environnement. Seules les propriétés accessibles via les API doivent être considérées.

3.1. *Propriétés locales*

Chaque appareil (électroménager) est pourvu d'une notice d'utilisation par son constructeur. Elle contient notamment des « prescriptions de sécurité et mises en gardes ». Ces prescriptions sont naturellement destinées à des utilisateurs humains, mais doivent entrer en considération lors de l'utilisation par un programme informatique. Ci-dessous, on rapporte deux prescriptions associées à une bouilloire électrique :

L1 Ne pas ouvrir le couvercle de la bouilloire lorsque l'eau bout (risque de brûlure).

L2 Ne pas allumer la bouilloire lorsqu'il n'y a pas assez d'eau dans le réservoir (risque de dysfonctionnement).

Un service coordonnant l'utilisation de cette bouilloire doit être tel (1) qu'il n'ouvre pas la bouilloire si l'eau bout et (2) qu'il ne doit pas allumer la bouilloire si le niveau d'eau est insuffisant.

Il existe des « prescriptions de sécurité et d'utilisation » qui concernent l'installation de l'appareil dans un environnement « adéquat », incluant la description du voltage et de la tension nécessaires, de la puissance de l'appareil, les fourchettes de température et d'humidité acceptables, etc.

3.2. *Propriétés globales*

Les propriétés locales concernent les propriétés correspondant à un appareil en particulier. Comme un service coordonne l'exécution de plusieurs appareils, certaines propriétés peuvent faire intervenir plusieurs appareils. On les appelle alors des « propriétés globales ». Par exemple, le service douche (S3) doit prévenir les risques de brûlure (propriété G1) ; le service préparation de la cuisine (S4) doit prévenir des risques d'intoxication par le monoxyde de carbone[1] (propriété G2).

G1 Le service S3 ne doit pas ouvrir le robinet d'eau de la douche si la température de l'eau au niveau du chauffe-eau est supérieure à 45 degrés.

G2 Le service S4 doit s'assurer que le ventilateur reste allumé tant que le robinet de gaz est ouvert.

Parmi ces propriétés, on peut aussi considérer l'expression du bon fonctionnement du service considéré. Ainsi, pendant l'exécution du service Home-Cinéma, les lumières doivent rester éteintes et les rideaux fermés.

3.3. *Propriétés issues de l'environnement*

Les services domotiques s'exécutent généralement au sein d'une habitation (ou assimilé). On peut donc considérer l'habitat comme faisant partie de l'environnement du système. Certaines propriétés en découlent. On les appelle « propriétés issues de l'environnement ». Il est possible d'extraire ces propriétés au travers de notices (on ne doit pas consommer plus d'électricité que ce qui est fixé au niveau du compteur : E1) ou du règlement de co-propriété (prévention des nuisances sonores E3). D'autres propriétés sont simplement issues du bon sens (risques incendie E2).

E1 La quantité maximale de courant utilisée ne doit pas excéder 30 ampères (maximum autorisé par le compteur).

E2 Les portes et fenêtres ne doivent pas rester verrouillées en cas d'incendie

E3 Pour éviter que le voisinage ne soit gêné par le bruit, éviter l'utilisation des appareils électroménagers (bruyants) entre 21 heures et 8 heures.

3.4. *Correction d'un service*

Soit s un service. Soit $App(s) = \{d_1, d_2, ..., d_n\}$ l'ensemble des appareils connectés au réseau utilisé par s. Et soit $LocalProp(d_i) = \{lp_{i1}, lp_{i2}, ..., lp_{im}\}$ l'en-

1. « Une insuffisance d'air ou une mauvaise évacuation des gaz brûlés peuvent entraîner la formation de monoxyde de carbone, un gaz inodore et très toxique », selon une recommandation de Gaz de France trouvée sur `http ://dolcevita.gazdefrance.fr/`.

semble des propriétés locales du d_i. L'ensemble des propriétés locales qui concerne le service s est défini par $LocalProp(s) = \cup_{d_i \in App(s)} LocalProp(d_i)$.

Soit $GlobalProp(s) = \{gp_1, gp_2, ..., gp_k\}$ l'ensemble des propriétés globale specifies pour le service s. Enfin soit $EnvProp(s) = \{ep_1, ep_2, ..., ep_l\}$ l'ensemble des propriétés de l'environnement dans lequel s est mis en œuvre.

On définit ainsi trois niveaux de correction.

– **Correction locale**. Un service s est *correct localement* si et seulement si s satisfait la conjonction des propriétés de $LocalProp(s)$.

– **Correction globale**. Un service s est *correct globalement* si et seulement si s la conjonction des propriétés de $GlobalProp(s)$.

– **Correction par rapport à l'environnement**. Un service s est *Correct par rapport à l'environnement* si et seulement si s satisfait la conjonction des propriétés de $EnvProp(s)$.

Un service s est *correct* si et seulement si s est correct localement, globalement et par rapport à son environnement.

4. Un modèle pour les services domotiques

Pour décrire puis valider un système domotique, nous nous appuyons sur un modèle objet (Fig. 2). Ce choix s'explique par le fait que l'équipe souhaite à moyen terme valider des implantation de systèmes domotiques grandeur nature, telles que décrites dans [NAK 05, LEE 05, NAK 06], et développées en Java.

La maison est considérée comme un objet à part entière. Elle est constituée (composée) d'appareils et de capteurs (électroménagers, lumières, ...). Chaque appareil a un état représenté par l'ensemble de ses attributs, et peut-être contrôlé via des méthodes publiques (formant son API). La maison est aussi composée de services. Chaque service est aussi modélisé par un objet, ayant un état et des méthodes parmi lesquels `callService` pour solliciter le service et `getServiceStatus` pour connaître son état.

4.1. *Les appareils*

Les appareils sont modélisés par une classe abstraite `Appliance`, elle même raffinée par des classes concrètes pour décrire chaque appareil. La classe abstraite `Appliance` regroupe les attributs et méthodes communs à tous les types d'appareils électriques. Elle est en particulier associée à une classe `Specification` qui contient de façon statique, les informations sur la nature de l'alimentation électrique, la puissance, la taille, les fourchettes de température et d'humidité ambiantes admissibles... Les méthodes de la classe abstraite

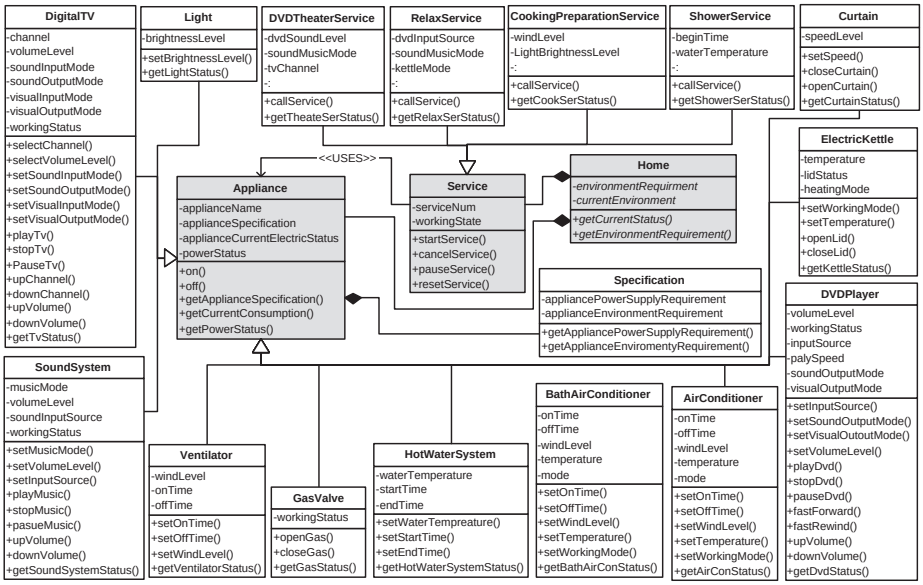

Figure 2. *Modèle objet pour décrire un système domotique*

`Appliance` sont les méthodes génériques pour allumer et éteindre les appareils électriques (`ON()`, `Off()`), pour connaître la consommation électrique courante (`getCurrentConsumption()`), et pour obtenir des informations sur la spécification (`getApplianceSpecification()`). Pour chaque appareil, les classes contrètes sont instanciées autant de fois que nécessaires. Ainsi, si la maison comporte 10 lampes, on définira 10 instances de la classe `Light` pour l'objet `Home`.

Les opérations spécifiques à chaque appareil sont décrites dans la sous-classe de `Appliance` correspondante. De telles méthodes correspondent à `AirConditioner.setWorkingMode("Cooling")`, `DVD.playDvd()`, ou `TV.selectChannel()`. Lorsqu'une méthode d'un appareil est appelée, l'état de l'appareil (via ses attributs) peut être modifié. La méthode `getTvStatus()` est utilisée pour connaître l'état interne d'un appareil.

4.2. *Les services*

Un objet « service » modélise un service domotique utilisant plusieurs appareils. Comme pour le cas des appareils, une classe abstraite `Service` est introduite. Elle décrit les méthodes communes à tous les services telles que `startService()` et `cancelService()`. Cette classe est concrétisée par une sous-classe pour chaque service spécifique. Pour réaliser le service Home-Cinéma, la

classe concrète `DVDTheaterService` utilise/commande mes objets `DigitalTV`, `DVDPlayer`, `SoundSystem`, `Light`, et `Curtain`.

4.3. *La maison*

La classe `Home` modélise la maison et définit les attributs environnementaux. De tels attributs correspondent à la consommation électrique, au niveau sonore, à la température, à la luminosité... Ils correspondent par exemple à l'état de capteurs disséminés dans la maison. Un modèle plus complexe pourrait faire apparaître les capteurs explicitement sur le même principe que les appareils (avec une classe abstraite, des classes concrètes et des instanciations). Mais pas soucis de simplicité ici, la mise à jour des attributs environnementaux est obtenue par appel de méthodes. Ainsi `Home.currentEnvironment.getTemperature()` permet d'obtenir la température ambiante.

5. Expression des propriétés dans le modèle

5.1. *Java Modeling Language*

Afin d'exprimer formellement les spécifications des services, on se propose d'utiliser JML (Java Modeling Language) [LEA 99, JML05]. JML est un langage conçu pour la spécification de programmes Java. JML permet de spécifier les programmes par l'expression de propriétés formelles sur les classes et leurs méthodes, sur le principe de l'approche « Design By Contract » [MEY 92]. Le cœur du langage JML est basé sur Java, avec quelques mots clef et constructions logiques supplémentaires [JML05, LEA 99]. Les annotations sont décrites au sein de commentaires (entre /*@ et @*/ ou sur une (fin de) ligne commençant par //@).

Les annotations JML reposent sur trois types d'assertions : les invariants, les pré-conditions et les post-conditions. Les invariants sont des propriétés qui doivent être vraies dans tous les états visibles. Un état visible correspond grossièrement à l'état initial et à l'état final de l'invocation d'une méthode. Une pré-condition d'une méthode (clause `requires`) indique que l'assertion doit être satisfaite avant l'exécution de la méthode, faute de quoi on ne peut garantir le résultat attendu après exécution de la méthode. Une post-condition d'une méthode (clause `ensures`) indique que l'assertion doit être satisfaite juste après l'exécution de la méthode (si les pré-conditions ont été respectées).

Parmi les autres mots clefs de JML, on trouve `\result` qui dénote la valeur retournée par une méthode (uniquement utilisable dans une post-condition) ; `\old(Expr)` correspondant à la valeur qu'avait `Expr` dans l'état initial de l'appel de la méthode ; `\forall` et `\exists` correspondant aux quantificateurs universels et existentiels.

Les annotations JML insérées dans le fichier source peuvent être compilées avec le compilateur JML `jmlc`. L'outil `jmlc` transforme les annotations JML en des assertions embarquées dans le code Java compilé [CHE 02]. Les assertions JML sont alors évaluées à l'exécution et comparées au comportement du programme. Si une assertion n'est pas vérifiée, une exception est générée, indiquant quel type d'assertion a été violée (invariant, pré ou post-condition).

Le code généré par `jmlc` peut être utilisé en combinaison avec l'outil JUNIT [JUn] lors d'un processus de test. JUNIT est particulièrement bien adapté pour exécuter (et suivre l'exécution) de test unitaires. D'autres outils sont aussi disponibles pour la peuve formelle de programmes (LOOP [BER 01], KRAKATOA [MEY 00] ou JACK [BUR 03]). Enfin, l'outil ESC/JAVA [FLA 02] est un outil qui permet d'identifier (et de corriger) des erreurs à l'aide d'une analyse statique.

En ce qui concerne notre système domotique, l'idée est bien sûr d'exprimer les propriétés locales, générales et issues de l'environnement sous la forme d'assertion au niveau des classes appareil, service et maison.

```
public class ElectricKettle{
    private /@spec_public@/ LidStatus lid;
    private /@spec_public@/ HeatingMode hstate ;

   /*@ public invariant (lid == 'open' && hstate != 'boiling') @*/

   /*@ requires hstate != 'boiling';
     @ ensures (lid == 'open' && hstate != 'boiling')
   */
    public openLid(){
      // code
   }
```

Figure 3. *Description JML de la propriété L1*

5.2. *Expression des propriétés locales*

Prenons l'exemple de la bouilloire électrique. Pour exprimer L1, il nous faut considérer les attributs `hstate` (indiquant si l'eau bout) et `lid` (indiquant si le couvercle est ouvert). On peut alors exprimer un invariant indiquant que l'on n'a jamais la situation où la bouilloire est ouverte avec de l'eau bouillante (Fig. 3). De plus, on peut spécifier explicitement que pour ouvrir la bouilloire (méthode `ElectricKettle.openLid()`), l'eau ne doit pas être en train de bouillir (pré-condition). Enfin, avec une post-condition, on s'assure que pendant l'exécution de la méthode, l'eau n'est pas devenue bouillante.

5.3. *Expression des propriétés globales*

Les propriétés globales s'expriment au niveau de la classe `Service` ou de ses sous-classes. Par exemple, la propriété G2 indique que le service S4 doit s'assurer que le ventilateur reste allumé tant que le robinet de gaz (`GasValve`) est ouvert. Cette propriété peut s'exprimer par l' invariant suivant :

```
/*@ public invariant (( GasValve.workingStatus == open) =>
Ventilator.powerStatus == ON) @*/
```

5.4. *Expression des propriétés issues de l'environnement*

Naturellement, les propriétés issues de l'environnement sont exprimées au niveau de la classe `Home`. Par exemple, pour décire la propriété E1 (la quantité maximale de courant utilisée ne doit pas excéder 30 ampères), on utilise la méthode `Home.currentEnvironment.getTotalConsumption()` qui retourne la quantité de courant actuellement utilisée au sein de la maison. On peut alors exprimer E1 sous la forme d'un invariant de la classe `Home` :

```
/*@ public invariant (currentEnvironment.getTotalConsumption()<=30) @*/
```

6. De la validation de services en utilisant JML

Cette section décrit la démarche de test mise en œuvre pour établir la correction des services. Le paragraphe 6.1 décrit de façon plus précise l'utilisation des assertions JML comme oracle. Le paragraphe suivant décrit quel processus de test a été mis en place.

6.1. *JML comme oracle pour le test*

Comme nous l'avons dit précédemment, JML peut être utilisé comme oracle pour le test, notamment en utilisant l'outil JUNIT. Durant l'exécution du système sous test, invariants, pre et post-conditions sont évalués en particulier au début et à la fin de l'exécution de chaque méthode.

Lorsqu'une opération est exécutée, trois cas peuvent se produire :

– Toutes les vérifications se sont déroulées correctement : le comportement de l'opération satisfait sa spécification pour les entrées choisies, dans l'état initial considéré. Le test retourne le verdict PASS.

– Une verification itermédiaire ou finale échoue. Cela traduit une inconsistance entre le comportement de l'opération et sa spécification. Autrement dit, l'implantation n'est pas conforme à sa spécification et le test retourne le verdict FAIL.

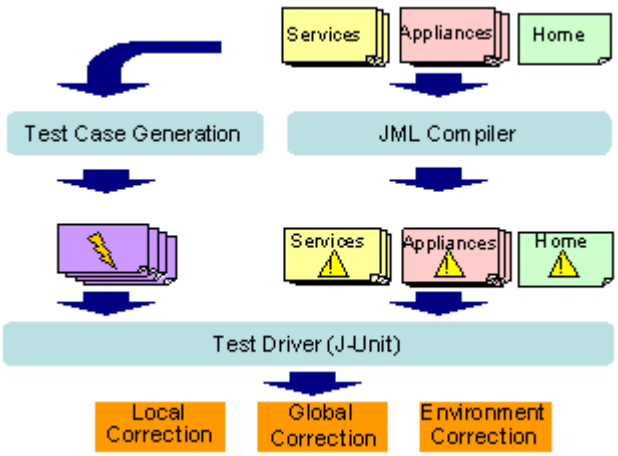

Figure 4. *Validation de services*

– Une vérification initiale (pre-condition) échoue : dans ce cas, exécuter le test jusqu'au bout n'apportera pas d'informations supplémentaires car le comportement observé est en dehors du comportement spécifié. Le test retourne le verdict INCONCLUSIVE.
Par exemple, $\sqrt{x}$ a comme pré-condition que x doit être positif. Ainsi, un test de cette méthode avec la valeur -1 délivera un verdict INCONCLUSIVE.

6.2. *Démarche de validation pour les services*

D'abord, nous procédons au test unitaire de l'implantation de chaque appareil. On s'assure que l'appel à chaque méthode (dans le cadre des préconditions) permet de préserver les invariants et de satisfaire les post-conditions.

Ensuite, nous procédons une phase de test où un appareil est « installé dans la maison ». Le but de cette étape est de s'assurer que l'utilisation du dit appareil n'est pas en contradiction avec les propriétés issues de l'environnement. En fait, de telles contradictions sont assez fréquentes. En effet, les propriétés issues de l'environnement traduisent à la fois des contraintes fortes (E2 - les portes ne doivent pas rester verrouillées en cas d'incendie) et des recommandations d'usage (E3 - pour le confort de tous, ne pas utiliser les appareils électoménagers entre 21h et 8h). Or il est assez facile de mettre le système dans un état où les « recommandations d'usage » peuvent être violées. D'une certaine façon, cette phase de test permet d'identifier les situations qu'il faudra prendre en compte lors de l'élaboration et/ou de la validation des services.

Vient ensuite la validation d'un service seul dans le cadre d'un environnement particulier. Il s'agit de s'assurer que le service respecte les propriétés locales, les propriétés issues de l'environnement et les propriétés globales. Diverses séquences sont produites en sollicitant et suspendant le service dans différentes situations. La complexité réside ici en la prise en compte de toutes les (du maximum de) situations possibles.
Cette étape conduit généralement à une complexification du service. Le code n'est plus simplement une suite d'appels telle que décrite Fig. 1. Les appels de méthodes doivent être précédés d'expressions conditionnelles s'assurant que les restrictions d'utilisation des appareils sont bien respectées. De plus, l'expression des propriétés globales d'un service doivent être raffinées pour prendre en compte les propriétés issues de l'environnement. Par exemple, le règlage du son des enceintes pour le service Home-Cinéma doit tenir compte de l'heure afin de s'ajuster automatiquement plus bas à 21h.

Enfin, on cherche à valider plusieurs services en même temps. Exécutés en même temps, les services sont amenés à interragir les uns avec les autres. Prenons l'exemple des services S1 et S2 (service Home-Cinéma et service de relaxation). Chacun utilise le lecteur de DVD et le climatiseur avec des réglages différents. Si le service Home-Cinéma est activé (`startService()`) alors que le service de relaxation est encore actif, alors les propriétés de globales de l'un des deux services peuvent ainsi être mises en défaut. L'interaction de services est une problématique complexe décrite dans [NAK 05] et largement couverte dans le cadre des services téléphoniques [BOU 94, CHE 95, KIM 98, CAL 00, AMY 03, REI 05].

7. Travaux connexes

Modélisation de services

Les services domotiques entrent dans le contexte plus large de l'informatique « ambiante/ubiquitaire ». Des exemples de services peuvent être trouvés dans [WEI 93, LAN 05]. Un autre exemple de réalisation de services domotiques peut se trouver dans [BOU 07a]. Le besoin en terme de qualité/sûreté de fonctionnement est souligné dans les précédents articles, mais aucune méthode générale n'est proposée.

Utilisation de JML pour valider du code Java

JML est un langage d'assertion pour Java. Ce n'est pas le seul. Depuis sa version 1.4 Java dispose d'un mecanisme d'assertion. Une assertion en Java est une condition booléenne pouvant être évaluée à l'exécution. Les options du compilateur Java permettent de choisir d'évaluer ou non les assertions à

l'exécution. Les assertions sont un mécanisme plus simple que JML et moins bien outillées.

Validation de services domotiques

La validation de services domotiques s'apparente à la validation de services téléphoniques, et en particulier en ce qui concerne le problème de la détection « d'interactions » entre services [BOU 94, CHE 95, KIM 98, CAL 00, AMY 03, REI 05]. De nombreux travaux ont été proposés pour détecter et résoudre les interactions de services. Dans le domaine des télécommunications, les approches de détection/résolution sont souvent classées en deux catégories « off-line » et « on-line » (ou à la volée). Nos travaux s'apparent plus à de la détection « off-line », puisque (1) nous raisonnons sur un modèle et que (2) nous recherchons des solutions pour ne pas faire appraître de conflits. Les stratégies de résolution de conflits que nous envisageons dans un premier temps est d'établir une priorité dans les services (comme cela a déjà implanté pour les services téléphoniques).

8. Conclusion

Dans ce papier, nous avons décrit un modèle simple de système domotique basé sur une conception objet et implanté en Java. Les objets élémentaires manipulés sont les appareils, les services et la maison. Puis, nous avons exprimé les besoins en terme de validation. Nous appelons « propriétés locales » les spécifications relatives à l'usage d'un appareil. Les « propriétés globales » sont les propriétés décrivant le comportement attendu d'un service. Ces propriétés concernent entre-autre l'utilisation combinée de plusieurs appareil. Enfin les « propriétés issues de l'environnement » sont souvent des restictions d'usage générales dans la maison.

Ces contraintes sont formalisées en JML (Java Modeling Language), sous la forme d'assertions de type invariant, pré et post-conditions. Ces assertions sont utilisées comme oracle, pendant les phases de test. La démarche de test consiste à solliciter successivement les appareils en dehors de tout contexte, puis dans le contexte particulier d'une maison. Ensuite, les services sont testés un par un, puis en combinaison.

La spécification et la validation de systèmes domotiques est difficile. La spécification nécessite une analyse approfondie et une formalisation des documents disponibles (mode d'emploi, règlements, ...) et des propriétés exprimant le bon sens mais non formulées explicitement. La validation nécessite d'examiner de nombreuses situations. Les premiers résultats sur notre modèle en terme de test nous ont conduit à décrire de façon plus précise les propriétés

globales et de rafiner les scénarios pour prendre en compte les restrictions d'usage issues de l'environnement et des appareils.

En terme de perspectives, il nous paraît intéressant de travailler selon deux directions. Tout d'abord, on peut détailler le modèle afin de prendre l'aspect dynamique d'un système domotique. En effet, un tel système ne peut rester figer. La « configuration » d'une maison change régulièrement : des appareils ou capteurs sont (des)installés et/ou tombent en panne. De même, les services sont introduits, modifiés ou supprimés au grè des besoins et des envies. Les services doivent aussi s'adapter aux changements dans la maison. L'expression des propriétés dans ce contexte reposera sur l'utilisation des quantificateurs existentiels et universels de JML.

Une seconde direction du travail consistera à se détacher progressivement du modèle, pour travailler directement sur les implantations décrites dans [NAK 05, LEE 05, NAK 06, BOU 07a]. Le traitement des exceptions JML au niveau de l'implantation sera un moyen de détecter et de résoudre dynamiquement des situations conflictuelles générées en particulier au niveau des services.

Remerciements. Les auteurs remercient le *Nara Institute of Science and Technology* et Egide avec son Programme d'Actions Intégrée (PAI-Sakura) pour leurs financements.

9. Bibliographie

[AMY 03] Amyot D., Logrippo L., Eds., *Feature Interactions in Telecommunications and Software Systems VII, June 11-13, 2003, Ottawa, Canada*, IOS Press, 2003.

[BER 01] van den Berg J., Jacobs B., « The LOOP Compiler for Java and JML », *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, vol. 2031 de *LNCS*, Springer, January 2001.

[BOU 94] Bouma L., Velthuijsen H., Eds., *Feature Interactions in Telecommunications Systems*, IOS Press, 1994.

[BOU 04] du Bousquet L., Ledru Y., Maury O., Oriat C., Lanet J.-L., « A case study in JML-based software validation (short paper) », *Automated Software Engineering (ASE)*, Linz, Austria, September 2004.

[BOU 07a] Bourcier J., Escoffier C., Lalanda P., « Implementing home-control applications on service platform », *4th IEEE Consumer Communications and Networking Conference (CCNC'07)*, Las Vegas, Nevada, USA, January 2007, IEEE.

[BOU 07b] du Bousquet L., Ledru Y., Lydie Dadeau F., Allouti F., « A Case Study in Matching Test and Proof Coverage », *3rd Workshop on Model-Based Testing (MBT), Satellite workshop of ETAPS 2007*, Braga, Portugal, March 2007.

[BUR 03] Burdy L., Requet A., Lanet J.-L., « Java applet correctness : a developer-oriented approach », *12th International FME Symposium*, Italy, September 2003.

[CAL 00] Calder M., Magill E. H., Eds., *Feature Interactions in Telecommunications and Software Systems VI, May 17-19, 2000, Glasgow, Scotland, UK*, IOS Press, 2000.

[CHE 95] Cheng K., Ohta T., Eds., *Feature Interactions in Telecommunications Systems III*, IOS Press, 1995.

[CHE 02] Cheon Y., Leavens G. T., « A Runtime Assertion Checker for the Java Modeling Language (JML) », Arabnia H. R., Mun Y., Eds., *International Conference on Software Engineering Research and Practice (SERP '02)*, Las Vegas, Nevada, June 2002, CSREA Press, p. 322–328.

[FLA 02] Flanagan C., Leino K. R. M., M. L., Nelson G., Saxe J. B., Stata R., « Extended static checking for Java », *Proc. of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, ACM Press, 2002, p. 234–245.

[JML05] « The JML Home Page », 2005, http ://www.jmlspecs.org.

[JUn] « JUnit », http ://www.junit.org.

[KIM 98] Kimbler K., Bouma L., Eds., *Feature Interactions in Telecommunications and Software Systems V*, IOS Press, 1998.

[LAN 05] Langheinrich M., Coroama V., Bohn J., Mattern F., « Living in a Smart Environment – Implications for the Coming Ubiquitous Information Society », *Telecommunications Review*, vol. 15, nº 1, 2005, p. 132–143.

[LEA 99] Leavens G., Baker A., Ruby C., « JML : A Notation for Detailed Design », Kilov H., Rumpe B., Simmonds I., Eds., *Behavioral Specifications of Businesses and Systems*, p. 175–188, Kluwer, 1999.

[LED 04] Ledru Y., du Bousquet L., Maury O., Bontron P., « Filtering TOBIAS Combinatorial Test Suites », *7th Int. Conf. FASE, Held as Part of ETAPS*, vol. 2984 de *LNCS*, Barcelona, Spain, 2004, Springer, p. 281-294.

[LEE 05] Leelaprute P., Tsuchiya T., Kikuno T., Nakamura M., Matsumoto K.-I., « Describing and Verifying Integrated Services of Home Network Systems », *12th Asia-Pacific Software Engineering Conference (APSEC'05*, December, Taipei, Taiwan 2005, IEEE Computer Society, p. 549-560.

[MEY 92] Meyer B., « Applying "Design by Contract" », *IEEE Computer*, vol. 25, nº 10, 1992, p. 40-51.

[MEY 00] Meyer J., Poetzsch-Heffter A., « An Architecture for Interactive Program Provers », *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 1785 de *LNCS*, Springer, 2000.

[NAK 05] Nakamura M., Igaki H., Matsumoto K.-I., « Feature Interactions in Integrated Services of Networked Home Appliances : An Object-Oriented Approach », *Feature Interactions in Telecommunications and Software Systems VIII, (ICFI'05)*, Leicester, UK, June 2005, IOS Press, p. 236-251.

[NAK 06] Nakamura M., Tanaka A., Igaki H., Tamada H., Matsumoto K.-I., « Adapting Legacy Home Appliances to Home Network Systems UsingWeb Services », *2006 IEEE International Conference on Web Services (ICWS'06)*, Chicago, Illinois, USA, September 2006, IEEE Computer Society, p. 849-858.

[REI 05] Reiff-Marganiec S., Ryan M., Eds., *Feature Interactions in Telecommunications and Software Systems VIII, ICFI'05, 28-30 June 2005, Leicester, UK*, IOS Press, 2005.

[WEI 93] Weiser M., « Some Computer Science Issues in Ubiquitous Computing. », *Commun. ACM*, vol. 36, nº 7, 1993, p. 74-84.