Approche de sélection d'architecture basée augmentation de désordre pour les Systèmes Collaboratifs Ubiquitaires

Imen Abdennadher
Université de Sfax, ReDCAD
B.P. 1173, 3038 Sfax, Tunisie
imen.abdennadher@redcad.org

Ismael Bouassida Rodriguez
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
Université de Sfax, ReDCAD, B.P. 1173, 3038 Sfax, Tunisia
bouassida@laas.fr

Mohamed Jmaiel
Université de Sfax, ReDCAD
B.P. 1173, 3038 Sfax, Tunisie
mohamed.jmaiel@enis.rnu.tn

Résumé

Dans notre travail, nous nous intéressons à l'adaptation des Systèmes Collaboratifs Ubiquitaires (SCU) aux changements de leur contexte, en choisissant la configuration architecturale la plus adéquate à la situation courante. Le framework "Framework for Adaptive Collaborative Ubiquitous Systems" (FACUS) [9] est l'un des frameworks intéressants qui assurent l'adaptation architecturale des SCU. FACUS est basé sur la sémantique et les grammaires de graphes afin de générer des configurations architecturales correctes par construction. Dans le contexte du framework FACUS, nous proposons une approche de sélection inspirée de la politique de sélection entropie, pour choisir la configuration architecturale la plus adéquate à la situation contextuelle de l'application. Notre approche prend en considération les changements dans la répartition des composants sur les dispositifs, entre la configuration architecturale courante (qui n'est plus adéquate au contexte courant) et une configuration architecturale candidate à la sélection. Nous appliquons notre approche à un cas d'étude "Smart Building". Notre cas d'étude décrit la collaboration entre des capteurs et des actionneurs, et vise la réduction de la consommation de l'énergie du bâtiment.

Mots clés

Systèmes Collaboratifs Ubiquitaires (SCU), la configuration architecturale la plus adéquate, "Framework for Adaptive Collaborative Ubiquitous Systems" (FACUS), politique de sélection, entropie

1. INTRODUCTION

Le concept de l'informatique ubiquitaire a été introduit par Mark Weiser en 1991 : "Les technologies les plus profondément enracinées sont les technologies invisibles. Elles s'intègrent dans la trame de la vie quotidienne jusqu'à ne plus pouvoir en être distinguées." [12]. L'informatique ubiquitaire désigne donc le fait que l'informatique est omniprésente, c'est-à-dire qu'elle est parfaitement intégrée dans les objets et les endroits de notre vie quotidienne, et ceci d'une manière tellement invisible qu'on ne s'aperçoit pas de sa présence. Sancho et al. [10] confirment que les environnements ubuiquitaires posent de nouveaux défits et possibilités pour les systèmes distribués, notamment pour les systèmes collaboratifs distribués. Les SCU décrivent des activités collaboratives dynamiques exécutées dans des environnements ubiquitaires, où le contexte change fréquemment. Ainsi, l'adaptation de ces systèmes dynamiques à leur contexte est hautement requise. Il est possible de distinguer deux types d'adaptation des systèmes : arhitecturale et comportementale. Nous nous focalisons sur les travaux proposant des solutions pour l'adaptation architecturale des SCU.

Becker et Giese [1] présentent une approche d'adaptation des systèmes auto-adaptatifs au moment de l'exécution. Cette adaptation définit trois niveaux : gestion des objectifs, gestion des changements et contrôle des composants. La modélisation des systèmes auto-adaptatifs est basée sur les techniques de transformation des graphes et les stéréotypes UML.

Padovitz et al. [7] présentent un modèle de contexte et une approche de raisonnement développée avec des concepts du modèle de l'espace des états, qui décrit le contexte et les situations comme des structures géométriques dans un espace multidimentionnel. Ils présentent aussi un algèbre de contexte basé sur le modèle, qui assure le raisonnement distribué en fusionnant et partionnant les modèles de contexte qui représentent différentes perspectives des entités sur l'objet du raisonnement. Les auteurs montrent que leur modélisation et leur approche de raisonnement facilitent l'exploitation des agents mobiles adaptatifs sensibles au contexte dans les environnements ubiquitaires.

Gui et al. [4] proposent un framework facilitant l'adaptation avec plusieurs préoccupations en utilisant des modules

d'adaptation composables et réutilisables. Les plans d'adaptation multiples sont générés à partir de différents modules d'adaptation. Puis, ils fusionnent ces plans et évitent les conflits entre eux en fournissant des supports pour la détection et la résolution des conflits. Les auteurs proposent un intergiciel basé sur le modèle de composants orienté-service et une stratégie nommée "Normalized context matching degree" pour évaluer et sélectionner les composants applicables à l'adaptation. Ils résolvent les conflits possibles, liés aux préoccupations d'adaptation multiples, en utilisant la sémantique des acteurs et les conditions du contexte.

Haesevoets et al. [5] proposent une solution, nommée Macodo, pour gérer la complexité de la conception des applications collaboratives. Macodo fournit un support centré sur l'architecture pour développer des collaborations entre services adaptatifs, en mettant l'accent sur les collaborations entre services qui se déroulent dans un environnement de collaboration limité. Macodo est composé de trois parties complémentaires : des abstractions pour modéliser les collaborations adaptatives, des vues architecturales et un support intergiciel avec preuve de concept.

Sancho [9] présente FACUS, un framework pour l'adaptation des systèmes collaboratifs aux changements de leur contexte. Le framework FACUS est basé sur une approche de modélisation architecturale multi-niveaux, permettant la séparation de la modélisation de l'architecture de l'application en des niveaux d'abstraction. Ceci facilite la gestion des problèmes du système en résolvant indépendamment les problèmes de chaque niveau. En outre, la séparation entre les niveaux de l'architecture assure une adaptation multi-niveaux, qui prend en considération les exigences de l'utilisateur dans les niveaux supérieurs et les contraintes imposées par les ressources disponibles dans les niveaux inférieurs. Pour ces raisons, nous adoptons ce framework et nous choisissons de l'améliorer. Afin d'assurer l'adaptation des SCU, le framework FACUS utilise une instance de l'approche de modélisation multi-niveaux, en modélisant l'architecture de l'application en trois niveaux : le niveau application, le niveau collaboration et le niveau intergiciel. Le dernier niveau (intergiciel) est basé sur une communication basée sur les événements (Event-Based Communication - EBC). Il inclut les composants qui peuvent être directement exécutés. Les composants EBC sont organisés en tois types : Gestionnaire de canal (Channel Manager - CM), Producteur d'événement (Event Producer - EP) et Consommateur d'événement (Event Consumer - EC).

La modélisation multi-niveaux assure une adaptation des systèmes basée sur deux actions : le raffinement et la sélection. Le raffinement permet de passer d'une configuration architecturale de niveau application vers une autre de niveau collaboration qui l'implante. Puis, le raffinement génère un ensemble de configurations architecturales de niveau intergiciel, à partir de la configuration architecturale de niveau collaboration. Ensuite, la procédure de sélection du framework FACUS permet de choisir la configuration architecturale la plus adaptée à la situation contextuelle courante dans le niveau intergiciel. Nous nous focalisons à la procédure de sélection du framework FACUS. Particulièrement, nous étudions les politiques de sélection utilisées par cette procédure de sélection, qui sont la distance et la dispersion. Nous améliorons le framework FACUS en ajoutant une politique de sélection inspirée de la politique entropie, appelée "Augmentation de désordre".

La suite de ce papier est organisée comme suit. La section 2 présente des travaux de recherche traitant les politiques de sélection. La section 3 présente la problématique. La section 4 présente un cas d'étude "Smart Building", utilisant le framework FACUS pour son adaptation. La section 5 présente la politique de sélection "Augmentation de désordre". Finalement, la section 6 présente la conclusion.

2. TRAVAUX TRAITANT LES POLITIQUES DE SÉLECTION

Le processus d'adaptation du framework FACUS est basé sur un algorithme de sélection. Cet algorithme prend en considération deux facteurs, qui sont les paramètres de contexte et les politiques de sélection. Nous nous focalisons sur le deuxième facteur, donc nous présentons les politiques distance et dispersion implantées dans l'algorithme de sélection du framework FACUS. En outre, nous présentons des travaux de recherche traitant les politiques de sélection.

2.1 La politique "Distance"

Dans le framework FACUS, la politique distance utilisée dans la procédure de sélection de la configuration architecturale la plus adaptée de niveau intergiciel permet d'effectuer le calcul de la valeur de distance entre la configuration architecturale candidate à la sélection et la configuration architecturale courante. La valeur de distance entre la configuration architecturale candidate et la configuration architecturale courante de niveau intergiciel correspond au nombre de composants de type CM qui sont déployés dans des dispositifs distincts de ces deux configurations architecturales. Les configurations architecturales candidates à la sélection ayant des valeurs de distance moins élevées ont plus de chance d'être sélectionnées.

Wolf et Holvoet [13] citent la politique distance comme une métrique pour évaluer la performance du système. Ils la définissent comme étant la différence entre l'état courant et l'état désiré du système.

2.2 La politique "Dispersion"

Dans le framework FACUS, la procédure de sélection de la configuration architecturale la plus adaptée de niveau intergiciel utilise la politique dispersion. Cette politique permet de calculer, pour chaque configuration architecturale candidate à la sélection, le nombre de dispositifs sur lequels nous avons des CM déployés. Les configurations architecturales candidates à la sélection ayant des valeurs de dispersion les plus élevées ont plus de probabilité d'être sélectionnées.

Beverly et al. [2] traitent le problème de sélection des points d'entraînement pour les algorithmes d'apprentissage. Ils visent assurer que leur ensemble d'entraînement est bien distribué convenablement afin de généraliser vers des prédictions aléatoires. Ils utilisent la politique dispersion, notamment la dispersion d'adresse, comme une métrique de distribution. Pour calculer la dispersion d'adresse, ils cherchent la différence numérique entre chaque addresse et les addresses suivantes les plus proches.

2.3 La politique "Moyenne"

Selon Wolf et Holvoet [13], la moyenne est une métrique à considérer, dans plusieurs cas, pour les propriétés du système dans son ensemble. Les auteurs jugent le système comme un groupe d'entités. Ainsi, le fait de mesurer une propriété du groupe dans son ensemble (partculièrement la moyenne) implique l'intégration des propriétés des entités dans une seule mesure. Les auteurs citent l'exemple de calcul du degré de connectivité dans les réseaux mobiles ad hoc, qui peut être mesuré en prenant la moyenne du nombre de connexions de chaque nœud vers les autres nœuds du réseau.

Kàroly et al. [6] ont crée un algorithme "Node Weight Computation" (NWC), basé sur la politique moyenne. L'algorithme NWC prend en considération cinq paramètres : la charge du CPU, la charge de la mémoire, le niveau de la batterie, la qualité du lien et la position qui représente le degré du nœud. Le calcul du poids de chaque nœud est effectué par le calcul de la moyenne des valeurs pondérées de ses paramètres de contexte. Basés sur les valeurs des poids des nœuds, Kàroly et al. proposent une solution pour la sélection des nœuds les plus performants qui joueront le rôle de serveurs de zone dans le réseau mobile ad hoc.

2.4 La politique "Entropie"

Philip [11] définit une métrique pour quantifier l'hétérogénéité dans les systèmes multi-agents. Ceci permet de partitionner les agents en un nombre fini de groupes ayant différentes capacités. La métrique hétérogénéité présentée par Philip utilise le calcul de l'entropie qui capture le désordre dans le système multi-agents en se basant sur la formule 1 :

$$E(p) = -\sum_{i=1}^{M} p_i log(p_i) \quad (1)$$

L'entier M représente le nombre d'espèces ou de groupes dans le système multi-agents et le nombre $p_i \in [0, 1]$ définit la probabilité qu'un agent choisi au hasard appartient au groupe i (avec i ∈ { 0, ..., M }).

Cioara et al. [3] se focalisent sur la gestion en temps réel de l'efficacité énergétique des centres de services. Pour réaliser ceci, ils utilisent un algorithme auto-adaptatif sensible au contexte. Les auteurs définissent le concept de l'*entropie de la situation de contexte du système*. Ce concept permet de mesurer le degré de désordre du système et de son environnement d'exécution.

Dans le travail de Wolf et Holvoet [13], la politique entropie est considérée comme une métrique pour évaluer la performance d'une solution. Elle est définie par la formule 2 :

$$E = \frac{-\sum_{n}^{N} p_n.log(p_n)}{logN} \quad (2)$$

où p_n représente la probabilité que l'état n, parmi N états possibles, apparaît. Le dénominateur (log N) permet de normaliser la valeur de l'entropie.

Parunak et Sven [8] présentent l'entropie comme étant une mesure de désordre de macro-niveau, définie par la formule 3 :

$$S = -\sum_{i} p_i log(p_i) \quad (3)$$

où i représente à chaque fois un état parmi les différents états possibles du système et p_i représente la probabilité que le système soit à l'état i considéré.

3. PROBLÉMATIQUE

A partir de l'étude des travaux de recherche sur les politiques de sélection, nous constatons que ces politiques sont multidisciplinaires et qu'elles sont utilisées dans différents domaines afin de résoudre divers problèmes.

Dans notre travail, nous nous intéressons à la procédure de sélection implantée dans le framework FACUS. Cette procédure permet de trouver la configuration architecturale la plus adaptée de niveau intergiciel. Particulièrement, nous nous focolisons sur l'utilisation des politiques de sélection dans cette procédure. Dans l'implantation courante de la procédure de sélection du framework FACUS, il y a seulement deux politiques de sélection, qui sont la distance et la dispersion. Nous jugeons que la procédure de sélection du framework FACUS a besoin d'être plus efficace pour pouvoir sélectionner la meilleure configuration architecturale dans le niveau intergiciel. Par conséquent, nous proposons d'autres politiques de sélection pour améliorer cette procédure de sélection. D'après l'étude que nous avons effectuée sur les travaux existants traitant les politiques de sélection, nous remarquons que plusieurs travaux de recherche utilisant les politiques distance et dispersion citent d'autres politiques comme la moyenne et l'entropie. Nous nous intéressons à la politique entropie afin de créer une nouvelle politique de sélection "Augmentation de désordre", que nous ajoutons à la procédure de sélection du framework FACUS.

4. CAS D'ÉTUDE : "SMART BUILDING"

Nous présentons le cas d'étude "Smart Building" pour lequel nous avons développé une application "Smart Building Application" (SBA) pour faire interagir les différents dispositifs. Cette application utilise FACUS pour son adaptation au contexte.

4.1 Présentation du cas d'étude "Smart Building"

La motivation de notre cas d'étude "Smart Building" est née du besoin de réduire la consommation de l'énergie d'un bâtiment. Comme il est présenté dans la figure 1, le bâtiment est composé d'un ensemble de salles, dont chacune comporte une variété de dispositifs de nature hétérogène et ayant différentes capacités logicielles et matérielles. Le bâtiment est capable de détecter les changements de son contexte et d'adapter son comportement à ces changements. Cette auto-adaptation du bâtiment nécessite une communication et une collaboration entre différents dispositifs (capteurs et actionneurs). Les capteurs assurent des observations (monitoring) du contexte. Les actionneurs répondent aux changements du contexte et envoient leurs états à une entité centrale.

Un gestionnaire du bâtiment ("Building Manager") centralise les données de l'application SBA. A ce "Building Manager", se connectent des entités se trouvant dans la salle :

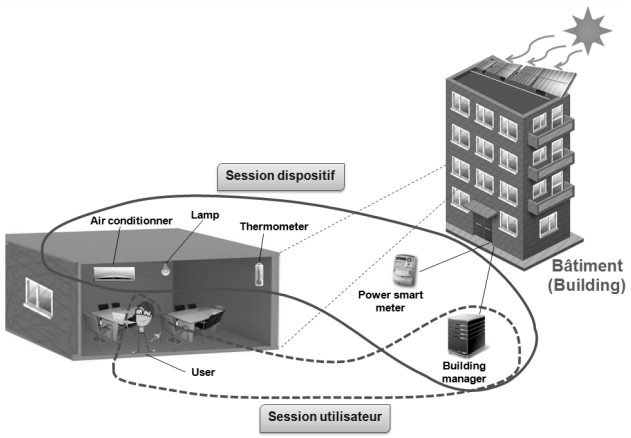

Figure 1: Le cas d'étude "Smart Building"

un climatiseur ("Air Conditioner"), un thermomètre ("Thermometer") et une lampe ("Lamp"). Un compteur d'énergie intelligent ("Power Smart Meter") gère la consommation et la production de l'énergie du bâtiment. Un utilisateur peut se connecter à notre application SBA et accéder à une interface depuis son PC, sa tablette, son smartphone, etc. L'application derrière cette interface s'appelle agent utilisateur ("User Agent"), elle peut se connecter au "Building Manager" et offrir différentes fonctionnalités à son utilisateur.

Nous considérons les six composants qui collaborent dans le "Smart Building" ("Building Manager", "User Agent", "Air Conditioner", "Thermometer", "Lamp" et "Power Smart Meter") comme des entités. Une entité est un nœud de collaboration qui participe à une activité collaborative. Elle joue un ou plusieurs rôles dans l'activité collaborative et elle est hébergée dans un dispositif [9].

Les entités qui collaborent au sein du cas d'étude "Smart Building" sont organisées suivant des sessions. L'organisation en sessions des entités du "Smart Building" est décrite dans la figure 1.

L'application SBA comprend deux sessions : une session utilisateur (présentée dans la figure 1 par un contour pointillé) et une session dispositif (présentée dans la figure 1 par un contour continu). La session utilisateur comporte les entités "Building Manager" et "User Agent". La session dispositif englobe les entités "Building Manager", "Air Conditioner", "Thermometer", "Lamp" et "Power Smart Meter". Cette organisation permet de préciser pour chaque entité l'ensemble des entités avec lesquelles elle peut collaborer et communiquer afin d'atteindre un objectif spécifique.

4.2 Architecture logicielle du cas d'étude "Smart Building"

La figure 2 représente la collaboration et la communication entre les différentes entités de l'application SBA ainsi que certaines données échangées entre ces entités. Sur la figure 2, l'entité "User Agent" envoie les valeurs requises par l'utilisateur de l'application à l'entité "Building Manager" (les valeurs de l'état et de l'intensité de la lampe et la valeur de la température du climatiseur demandées). Les entités "Lamp", "Air Conditioner", "Thermometer" et "Power Smart Meter" envoient périodiquement leurs valeurs (l'état et l'intensité de la lampe, les valeurs de consommation et de production de l'énergie, la valeur de la température mesurée par le thermomètre, la valeur de la température du climatiseur, les périodes d'envoi et de réception des différents valeurs des entités) à l'entité "Building Manager". Cette dernière peut envoyer des requêtes de modification des valeurs aux entités "Lamp" et "Air Conditioner".

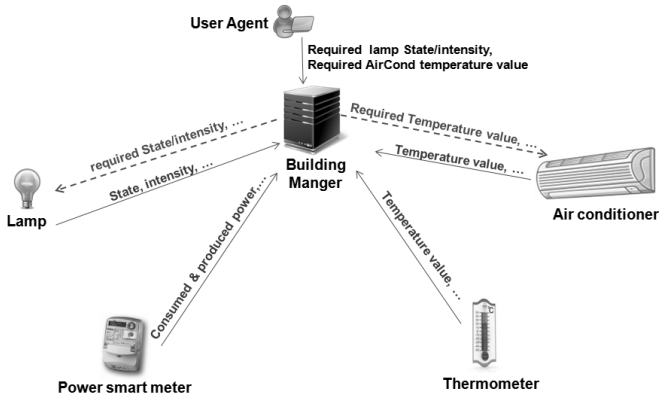

Figure 2: Architecture logicielle du "Smart Building"

Comme nous avons indiqué dans la section 4.1, l'application SBA comporte une session utilisateur et une session dispositif.

Dans la session utilisateur, les flux de données entre les deux entités "Building Manager" et "User Agent" sont l'ensemble des exigences de l'utilisateur (l'état de la lampe requis, l'intensité de la lampe requise, la valeur de la température du climatiseur requise) et les réponses à ces exigences.

Dans la session dispositif, nous avons deux types de flux de données :

- Les flux allant des entités "Air Conditioner", "Thermometer", "Lamp" et "Power Smart Meter" vers le "Building Manager" : représentent les valeurs des caractéristiques de ces quatre entités, envoyées vers le gestionnaire du bâtiment.
- Les flux allant du "Building Manager" vers les entités "Air Conditioner", "Thermometer", "Lamp" et "Power Smart Meter" : représentent les requêtes d'interrogation et d'ajustement des valeurs des caractéristiques des quatre entités ("Air Conditioner", "Thermometer", "Lamp" et "Power Smart Meter"), ainsi que les valeurs des propriétés des deux entités ("Air Conditioner" et "Lamp") demandées par le "Building Manager".

L'entité "Thermometer" joue le rôle d'un capteur en mesurant régulièremenet la valeur de la température de l'environnement. Cette valeur est envoyée ensuite à l'entité "Building Manager".

L'entité "Power Smart Meter" est reliée au réseau électrique publique pour y injecter le surplus de production et pomper l'énergie si nécessaire. Elle calcule les valeurs de l'énergie

produite et de l'énergie consommée et les envoie à l'entité "Building Manager".

Les deux entités "Air Conditioner" et "Lamp" envoient régulièrement leurs états à l'entité "Building Manager". De plus, elles jouent le rôle d'actionneurs. Elles subissent des ajustements des valeurs de leurs caractéristiques suite aux ordres envoyés par l'entité "Building Manager". Par exemple, l'entité "Building Manager" peut envoyer une rêquete d'extinction de la lampe, et par la suite, l'entité "Lamp" qui reçoit cette rêquete change son état de "allumée" vers "éteinte".

Les entités de l'application SBA fonctionnent suivant deux modes :
- Le mode passif : l'entité envoie les données à l'ensemble des entités qui l'ont interrogée.
- Le mode actif : l'entité décide d'envoyer (périodiquement ou à un moment donné) des données aux entités avec lesquelles elle collabore.

5. POLITIQUE DE SÉLECTION "AUGMENTATION DE DÉSORDRE"

Notre objectif est d'étendre la procédure de sélection implantée dans le framework FACUS. Nous nous sommes inspirés de la politique entropie pour définir une nouvelle politique de sélection que nous appelons "Augmentation de désordre". Dans cette section, nous définissons cette politique et comment elle peut être utilisée dans la procédure de sélection du framework FACUS.

5.1 Définition et principe de la politique "Augmentation de désordre"

Avant d'aborder notre politique, nous rappelons la politique entropie, qui permet de caractériser le désordre à l'intérieur d'un système. L'entropie est définie par la formule 4 :

$$S = -\sum_{i=1}^{N} p_i log(p_i) \quad (4)$$

où N représente le nombre d'états possibles du système, i représente à chaque fois un état parmi les N états possibles du système et p_i représente la probabilité que le système soit à l'état i [8]. Ainsi, la mesure de l'entropie est calculée pour tous les états possibles du système.

Contrairement à la politique entropie, notre politique "Augmentation de désordre", ne caractérise pas le désordre du système. Cependant, elle s'intéresse particulièrement à calculer l'augmentation de désordre à l'intérieur du système, à un état considéré. Elle est définie par la formule 5 :

$$S = -\sum_{i=1}^{N} p_i log(p_i) \quad (5)$$

où N représente le nombre de changements dans l'état considéré du système, i représente le i^{ème} changement parmi les N changements dans l'état considéré et p_i représente la probabilité que le changement i est réalisé.

Dans notre travail, un état correspond à une configuration architecturale candidate à la sélection de niveau intergiciel appelée A_i. Un changement d'état correspond à un changement au niveau de la configuration architecturale A_i. Le modèle de communication implanté dans le framework FACUS est le modèle EBC, qui comporte les composants CM, EP et EC. Les changements sont calculés en comparant les emplacements des CM dans la configuration architecturale courante par rapport à leurs emplacements dans chacune des configurations architecturales candidates à la sélection.

Un changement au niveau d'une configuration architecturale candidate à la sélection, calculé par rapport à la configuration architecturale courante, peut être :
- Un ajout de CM : le CM considéré n'est pas déployé dans la configuration architecturale courante, mais il sera déployé dans la configuration architecturale candidate à la sélection si cette dernière est retenue.
- Une suppression de CM : le CM considéré est déployé dans la configuration architecturale courante, mais il ne sera pas déployé dans la configuration architecturale candidate à la sélection si cette dernière est retenue. Dans le cas de suppression de CM, la valeur de la politique "Augmentation de désordre" reste inchangée.
- Une migration de CM : le CM considéré sera déployé dans la configuration architecturale candidate (si elle est retenue) sur un dispositif différent de celui sur lequel il a été déployé dans la configuration architecturale courante.

5.2 Définition des probabilités de migration et d'addition des composants CM

La probabilité de migration des CM entre les dispositifs des configurations architecturales de niveau intergiciel dépend au moins de deux facteurs :
- La situation de contexte du dispositif dans lequel le CM sera déployé : Plus le dispositif est loin de l'état critique (c'est-à-dire plus ses paramètres de contexte dépasssent les seuils avec des valeurs importantes), plus la probabilité de migration des CM vers ce dispositif est élevée.
- La structure interne du CM : En fait, le CM est un composant qui peut être composite (c'est-à-dire qui peut englober en lui même un ensemble de composants). Plus la structure interne du CM est simple, plus nous économisons les coûts de transfert, et donc plus la probabilité de migration de ce CM est élevée.

La probabilité d'ajout de CM dépend de la situation de contexte du dispositif dans lequel le CM sera déployé.

5.3 Algorithme de calcul de la politique "Augmentation de désordre"

L'algorithme de calcul de la politique "Augmentation de désordre" d'une configuration architecturale de niveau intergiciel est détaillée dans Table 1. Il prend comme paramètre d'entrée une configuration architecturale candidate à la sélection de niveau intergiciel (A_1), et retourne la valeur de la politique "Augmentation de désordre" (AD) de A_1 (Table 1 - ligne 3). AD vaut zéro si la configuration architecturale courante A_0 et la configuration architecturale candidate A_1 sont les mêmes. AD est strictement positive s'il y a addition ou migration de CM dans A_1 en la comparant à A_0.

Pour chaque CM appartenant à la configuration architecturale courante (A_0) ou à la configuration architecturale candidate à la sélection (A_1) (Table 1 - ligne 8) :

- S'il s'agit d'un ajout de CM (Table 1 - ligne 9), l'opposé de

Table 1: Algorithme de calcul de la politique "Augmentation de désordre" d'une configuration architecturale de niveau intergiciel

```
1 Augmentation_Desordre(A1){
2 Soit A0 la configuration architecturale courante
3 Soit AD = 0
4 Soit d1 le dipositif où CM est deployé dans A1
5 Soit d0 le dipositif où CM est deployé dans A0
6 Soit P_CM:d1 : la probabilité que CM est deployé
  dans d1
7 Soit P_CM:d0→d1 : la probabilité que CM migre
  de d0 vers d1
8 pour CM ∈ {A0 ∪ A1}
9  si (CM ∉ A0 and CM ∈ A1)
10  AD = AD − P_CM:d1 log(P_CM:d1)
11  sinon si (CM ∈ A0 and CM ∈ A1)
12   si (d0 ≠ d1)
13    AD = AD − P_CM:d0→d1 log(P_CM:d0→d1)
14   fin si
15  fin si
16 fin pour
17 retourner AD
18 }
```

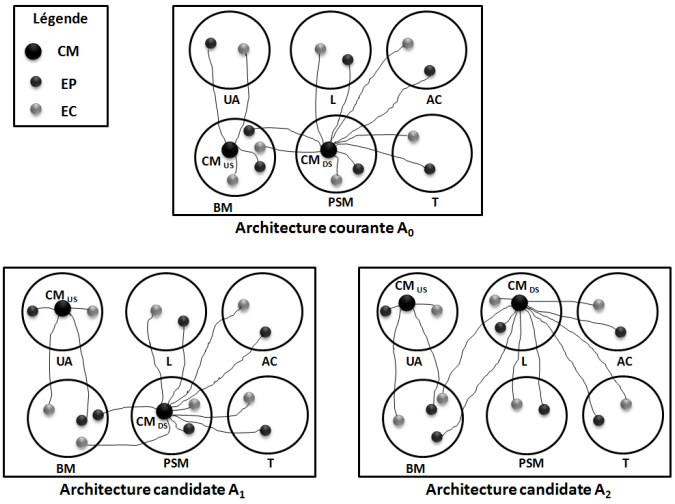

Figure 3: Exemple de configurations architecturales de niveau intergiciel

la pobabilité d'ajout de ce CM multupliée par son log (- $P_{CM:d_1} log(P_{CM:d_1})$) est ajoutée à AD (Table 1 - ligne 10).

- Sinon, s'il s'agit d'une migration de CM (Table 1 - lignes 11 et 12), l'opposé de la probabilité de migration de ce CM multupliée par son log (- $P_{CM:d_0 \to d_1} log(P_{CM:d_0 \to d_1})$) est ajoutée à AD (Table 1 - ligne 13).

5.4 Exemple de calcul de la politique "Augmentation de désordre"

Dans cette section nous présentons un exemple de calcul de la politique "Augmentation de désordre" dans le contexte de notre cas d'étude "Smart Building". La figure 3 expose la configuration architecturale courante de niveau intergiciel (A_0) et deux exemples de configurations architecturales candidates à la sélection (A_1 et A_2).

Dans chaque configuration architecturale, nous avons présenté les dispositifs "Building Manager" (BM), "User agent" (UA), "Lamp" (L), "Air Conditioner" (AC), "Thermometer" (T) et "Power Smart Meter" (PSM), présentés dans la figure 3 par des cercles. Dans ces dispositifs sont déployés les composants EBC : CM, EP et EC.

Dans la figure 3, CM_{US} représente le CM qui gère la session utilisateur et CM_{DS} représente le CM qui gère la session dispositif.

La configuration architecturale courante A_0 n'est plus adaptée à la situation de contexte courante. La procédure de sélection du framework FACUS choisit, parmi les configurations architecturales candidates à la sélection, celle qui est la plus adaptée au contexte courant. La procédure de sélection peut utiliser les valeurs de la politique "Augmentation de désordre" relatives aux configurations architecturales candidates à la sélection A_1 et A_2 pour sélectionner l'une de ces deux configurations architecturales.

Afin de calculer les valeurs de la politique "Augmentation de désordre", l'utilisateur du framework FACUS fournit les probabilités suivantes :

- la probabilité de migration de CM_{US} du dispositif BM vers le dispositif UA :
  $P_{BM \to UA}\ (CM_{US}) = 0{,}3$
- la probabilité de migration de CM_{DS} du dispositif PSM vers le dispositif L :
  $P_{PSM \to L}\ (CM_{DS}) = 0{,}5$

En appliquant l'algorithme de calcul de la politique "Augmentation de désordre", les résultats suivants sont obtenus :

- La valeur de la politique "Augmentation de désordre" pour l'architecture A_1 est :
  AD(A_1) = - 0,3 log (0,3) = 0,15
- La valeur de la politique "Augmentation de désordre" pour l'architecture A_2 est :
  AD(A_2) = - 0,3 log (0,3) - 0,5 log (0,5) = 0,3

Ainsi, la configuration architecturale choisie comme la plus adéquate au contexte est A_1 puisqu'elle représente la valeur la moins élevée de la politique "Augmentation de désordre" (0,15 < 0,3).

6. CONCLUSION

Dans ce papier, nous nous intéressons au processus d'adaptation du framework FACUS. Particulièrement, nous nous focalisons sur la procédure de sélection de la configuration architecturale la plus adaptée de niveau intergiciel. Nous améliorons cette procédure de sélection en ajoutant une politique de sélection "Augmentation de désordre". Nous concevons et implantons un cas d'étude "Smart Building", utilisant le framework FACUS pour adapter son comportement aux changements de son contexte.

Comme perspective à ce travail, nous visons évaluer notre approche en appliquant plus de tests de déploiement à notre cas d'étude "Smart Building". Nous visons aussi améliorer le cas d'étude en considérant d'autres dispositifs, comme des panneaux photovoltaïques, des capteurs de présence, etc.

7. REMERCIEMENT

Ce travail est supporté par le projet ITEA2 A2NETS (Autonomic Services in M2M Networks) [1]

8. RÉFÉRENCES

[1] B. Becker and H. Giese. Modeling of correct self-adaptive systems : A graph transformation system based approach. In *Proceedings of the 5th International Conference on Soft Computing As Transdisciplinary Science and Technology*, pages 508–516, 2008.

[2] R. Beverly, K. Sollins, and A. Berger. Svm learning of ip address structure for latency prediction. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, pages 299–304, 2006.

[3] T. Cioara, I. Anghel, I. Salomie, G. Copil, D. Moldovan, and B. Pernici. A context aware self-adapting algorithm for managing the energy efficiency of it service centres. *Ubiquitous Computing and Communication Journal*, 6 :619–630, 2011.

[4] N. Gui, V. D. Florio, and T. Holvoet. Transformer : an adaptation framework supporting contextual adaptation behavior composition. *Softw., Pract. Exper.*, 43 :937–967, 2013.

[5] R. Haesevoets, D. Weyns, and T. Holvoet. Architecture-centric support for adaptive service collaborations. *ACM Trans. Softw. Eng. Methodol.*, 23 :2, 2014.

[6] F. Kàroly, H. Theus, P. Bernhard, and R. Lukas. Nwc : Node weight computation in manets. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 1059–1064, 2007.

[7] A. Padovitz, S. Loke, and A. Zaslavsky. Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on*, 38 :729–742, 2008.

[8] V. D. Parunak and B. Sven. Entropy and self-organization in multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, pages 124–130, 2001.

[9] G. Sancho. *Adaptation d'architectures logicielles collaboratives dans les environnements ubiquitaires. Contribution à l'interopérabilité par la sémantique.* PhD thesis, Université Toulouse 1 Capitole (UT1 Capitole), Newark, December 2010.

[10] G. Sancho, I. B. Rodriguez, T. Villemur, and S. Tazi. What about collaboration in ubiquitous environments ? In *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on*, pages 143–150, 2010.

[11] P. Twu. *Control of multi-agent networks : from network design to decentralized coordination.* PhD thesis, the School of Electrical and Computer Engineering, 2012.

[12] M. Weiser. The computer for the 21st century. *Scientific American*, 265 :66–75, January 1991.

[13] T. D. Wolf and T. Holvoet. Evaluation and comparison of decentralised autonomic computing solutions. Technical report, 2006.

1. `https://a2nets.erve.vtt.fi/`