Conférence francophone sur les Architectures Logicielles (CAL'2016)

Besançon, France – Juin 7-8, 2016

http://www.cal-conference.fr

CAL'2016

Les articles acceptés

Model migration : reconstruction of metamodel evolution using an intelligent reasoning

Fouzia Anguel[1], Abdelkrim Amirat[2], Nora Bounour[3]

1. *Chadli Bendjedid university*
BP 73, El Tarf, 36 100, Algeria.
LISCO laboratory, Badji Mokhtar university
B.P.12, Annaba, 23000, Algeria.
fanguel@yahoo.fr

2. *LiM laboratory, Mohamed Cherif Messaadia university*
Souk Ahras,41000, Algeria.
abdelkrim.amirat@yahoo.com

3. *LISCO laboratory, Badji Mokhtar university*
B.P.12, Annaba, 23000, Algeria.
nora_bounour@yahoo.fr

RÉSUMÉ. L'évolution des métamodèles affecte tous le reste des artefacts qui en dépendent, à savoir les modèles ou les règles de transformation. Dans cet article, nous proposons une approche pour l'adaptation des modèles à l'évolution de leurs métamodèles (co-évolution). Le processus de co-évolution défini se compose de quatre phases permettant l'identification des évolutions à posteriori sous forme de différences, la reconstruction de ces différences en opérations d'évolution atomiques et composites, la détermination de la solution de migration des modèles et finalement, l'adaptation assistée des modèles à la nouvelle version du métamodèle. Cette approche intègre également une activité de formalisation d'un catalogue d'opérateurs d'évolution en utilisant la programmation logique ainsi que l'utilisation d'un mécanisme de raisonnement intelligent pour inférer les changements composites.

ABSTRACT. Metamodels evolution affects the rest of their dependent artifacts i.e. models or transformation rules. In this paper we propose an approach to adapt models to their evolved metamodels (i.e co-evolution). The defined co-evolution process consists of four phases which allow the identification of evolution as a set of differences, the reconstruction of these differences on atomic and compound evolution operations, the determination of models migration solutions and finally an assisted adaptation of models to the new metamodel version. This approach also incorporates a formalization activity of a catalog of evolution operators using logic programming and the use of an intelligent reasoning mechanism for inferring composite changes.

MOTS-CLES : Migration de modèle, Changement de métamodèle, Opérateur primitif, Opérateur composite, Programmation Logique, Modèle de différence.

KEYWORDS: Model migration, Metamodel change, Primitive operator, Composite operator, Logic Programming, Difference model.

1. Introduction

Evolution of software systems is a rapidly increasing need due to influence of technological innovation. The Model-Driven Engineering paradigm (MDE) (Bézivin, 2005) is not beyond that need; metamodels and domain-specific languages are key artifacts in MDE, as they are used to define syntax and semantics of domain models. Thus, metamodels are not created once and never changed again, but are in rather continuous evolution (Favre, 2003). The evolution of metamodels is relatively a broad problem because many artifacts are related to it and might be impacted by metamodel changes. Therefore metamodel evolution may require the migration of their related artifacts. In this paper we tackle the problem of model migration. In the literature, three general approaches to migrate models exist (Rose et al, 2009): manual, state-based and operator-based approaches. Manual approaches are tedious and error prone. State-based approaches also called difference-based approaches allow synthesizing a model migration based on the difference between two metamodel versions. In contrast, operator-based approaches allow capturing the intended model migration already when changing the metamodel (Herrmannsdörfer and Wachsmuth, 2014). We opted for a hybrid approach using at the same time matching techniques and operators to co-evolve models and metamodels and also applying an intelligent reasoning. Unlike existing approaches, our approach is oriented user and it is based on three principal axes. The first one is the formalization in Logic Programming (LP) of metamodels, changes and a library of atomic and composite operators. The second one is the reconstruction of evolution operations using an inference engine and the third one is the adaptation of models. User decisions are included during metamodel and model co-evolution process.

The paper is structured as follows. Section 2 describes metamodel and model co-evolution problem. Section 3 discusses existing works. Section 4 explains our proposed approach for solving the model co-evolution problem. In section 5, we give implementation guidelines and we introduce a running example to show applicability of our approach. Finally, we conclude the paper in section 6.

2. Metamodel and model co-evolution

In general MDE consists of the following two main artifacts (Garcès, 2009): modeling languages, which are used to describe a set of models and model transformation rules which are used to translate models represented in one language into models represented in another language. The abstract syntax of a modeling language is described by a metamodel. Metamodel is a model that defines concepts for expressing models (Bézivin, 2005). Models enable developers to reason about a

system at a higher level of abstraction. Analogously, metamodels are defined by concepts described as meta-metamodels. This hierarchy is specified by the Object Management Group (OMG) in four-layer modeling architecture (Bézivin, 2005). Metamodels can be expressed in various meta-modeling formalisms. Well-known examples are the Meta Object Facility (MOF) (OMG, 2015b). MOF is a standard; it shares many common modeling elements with UML. In this paper we focus only on the core meta-modeling constructs of MOF that are interesting in metamodels and models co-evolution.

In fact, metamodels undergo complex evolutions during their life cycles (Favre, 2003). As a consequence, when a metamodel is modified, models conforming to this metamodel need to be migrated in such a way they conform to the modified version. In the literature, this problem is referred to as metamodel evolution and model co-evolution (Wachsmuth, 2007), model adaptation (Garcès et al, 2009) or model migration (Rose et al, 2010).

Metamodel evolution is represented by a set of metamodel changes. A number of works proposed the classification of metamodel changes according to their corrupting effects on models. In (Gruschko et al, 2007) metamodel changes are grouped on three categories:

- not breaking changes, changes occurring in the metamodel don't break models conformance to the metamodel;
- breaking and resolvable changes, changes occurring in the metamodel do break models conformance, but can be automatically resolved;
- breaking and non-resolvable changes, changes do break the models and cannot be automatically resolved and user intervention is required.

However, a uniform formalization of metamodel evolution is still lacking (Rose et al, 2009). The relation between metamodel and model changes should be formalized in order to allow reasoning about the correctness of model migration definitions. This work proposes a formal approach to metamodel and model co-evolution which addresses this challenge.

3. Related works

Currently, there are several approaches that support this problem; Rose et al (2009) propose a classification of model migration approaches. That highlights three ways to identify needed model updates: 1) manually, 2) based on operators, and 3) state-based approaches. Manual approaches, like (Sprinkle, 2004; Narayanan, 2009; Rose, 2007) provide transformation languages to manually specify the model migration. In operators based approaches (Wachsmuth, 2007; Herrmannsdörfer, 2010) metamodels changes are defined in terms of co-evolutionary operators. Those operators define conjointly the evolution on the metamodel and encapsulate model migration. Wachsmuth (2007) shows an operation-based tool prototype called ECORALL which provide a library of reusable coupled operations as a basis for automatic metamodel evolution. Herrmansdoerfer developed COPE which explicitly records the history of the metamodel as a sequence of changes and allows attaching

information on how to migrate models (Herrmannsdörfer, 2010). To increase expressiveness, COPE extends the approach proposed in (Wachsmuth, 2007) with a means to specify a custom model migration for a recorded metamodel adaptation. Finally, state-based approaches use metamodel matching where versions of metamodels are compared and differences between them are used to semi automatically derive a transformation that expresses models updates. In (Gruschko et al, 2007) primitive metamodel changes are classed into not-breaking, breaking resolvable and breaking non-resolvable and envision to automatically detect a model migration for breaking resolvable changes. Moreover, Cicchetti et al have proposed capturing differences between models through the use of a difference metamodel and present a tool prototype that supports model evolution for primitive changes and compound changes (Cicchetti, 2008). In (Garcès, 2009) authors developed an EMF-based matching language that supports customization of the matching process and adding new matching patterns.

After this brief survey of existing approaches to metamodel and model co-evolution, we can discern that while manual approaches foster correctness of the model migration, they also require the most effort. However matching approaches try to completely automate the building process, they may not always lead to a correct model and there is not yet a mature approach in this category (Herrmannsdörfer and Wachsmuth, 2014). Operators based approaches lead to a correct migration by means of recording evolution trace at the same time and reduce the effort by reusing coupled operations. However, they also require integration into the editor for the metamodel.

In fact, analysis of advantages and disadvantages of existing approaches helps us to motivate our approach targeting to adapt models to their evolved metamodels. In this work, we propose an alternative solution where we use a mixture of state-based and operator-based approaches to adapt models to metamodel evolution. Our approach goes beyond existing work in three directions.

Firstly, Existing approaches do not differentiate the role of model designer (i.e. metamodel user) and metamodel designer. We notice a strong involvement of model designer in evolution process, although he is not always connoisseur of metamodel design and cannot always comprehend the intention behind the evolution process. Our proposed approach is oriented to model designer (i.e. user) and aims to provide to users a tool support to accomplish successfully the migration of their models without participating to metamodel evolution process. The objective is to reduce as much as possible the different interventions of the model designer and to guide the co-evolution process. This is why; we consider that metamodel changes are calculated a posteriori as in state-based approach.

Secondly, since operator based approaches gave good results (Herrmannsdörfer and Wachsmuth, 2014) and Specially COPE approach with its library covering a large set of operators (Herrmannsdörfer et al, 2011). We envisage using operators in our approach, however we have find that Cope approach is oriented implementation. In Cope, evolutions are described by coded functions to be executed and operators don't have a formal definition and are not documented. In contrast, our approach

proposes a formalization of a set of atomic and compound operators in order to use them more efficiently and to facilitate eventual library extension. The proposed library provides a high degree of expressiveness to this approach by adding logical structure.

Thirdly, correct evolution management needs considering both atomic and compound evolution operations. Most of existing approaches handle atomic ones (Rose et al, 2009). In our proposal the novelty is the use of an intelligent mechanism to reason about evolution operations in order to reconstruct compound operations and therefore to got an evolution scenario. Currently to the best of our knowledge there is no approach that uses an intelligent reasoning to resolve metamodel and model co-evolution problem.

4. Proposed approach

In this section we describe our proposal to perform the co-evolution of models with their evolved metamodels. The overall evolution and co-evolution process is presented in Figure 1. Our approach is hybrid and it imports techniques from state-based and operator based approaches and uses also a reasoning mechanism from artificial intelligence. It contains four phases: 1)-detection of changes, 2)-reconstruction and validation of evolution scenario, 3)-determination of migration scenario and 4)-running of model migration. In the first step; differences between two metamodel versions (MM1, MM2) need to be determined. In the second step we reconstruct atomic evolution operations from the difference model (MM2-MM1) then we use an inference engine to generate evolution scenarios by assembling atomic operations in possible compound ones; in the third step we explore a library of operators (i.e. Knowledge base "KB") to obtain different migration procedures, which will be assembled to constitute migration scenario. In the last step the migration scenario will be applied over a specific model M1 conforming to the old version in order to obtain a new model M2 conforming to the newer metamodel version. During this step users decide how the model will change based on the possible alternatives solutions. With this co-evolution process, we define a preliminary phase to encode metamodels, changes and a library of operators using logic programming formalism. Resulted knowledge base is the core of the proposed co-evolution process.

4.1. Preliminary phase

This phase consists on preparing environment to execute proposed co-evolution process. It is performed through three steps. Firstly, we propose a formalization of MOF based metamodels and models in LP. Secondly, we propose an encoding method to represent the set of possible metamodel changes using predicates. Thirdly, we define in LP evolution operators and categorize them in primitive and compound operators.

6

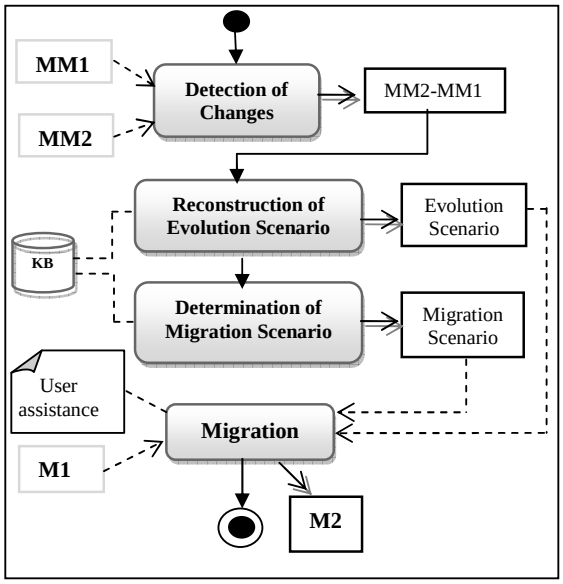

Figure 1. Overview of co-evolution process

4.1.1. Encoding MOF metamodels in LP

To reason about properties of metamodels and their evolution, a textual or a graphical representation is often not sufficient. Thus, we provide a more formal representation of metamodels in terms of predicates. Therefore, as each model is an instance of the metamodel, a formalized model can be deduced. To be an efficient approach, the selected language must allow all the MOF metamodels features to be expressed and must offer an analysis capability to reason about evolution of MOF metamodels and their models. In this step we start by analyzing MOF metamodels to identify evolutionary elements with their evolution parameters. However, in this paper we focus only on the core metamodeling constructs that are interesting for models migration. We leave out constructs which cannot be instantiated in models as packages, annotations, derived features, and operations. Because LP is based on facts and considering modeling hierarchy, we define here a format to represent constructs called meta-facts. The instantiation of a meta-fact is a fact which consists in giving constant values to the parameters of this meta-fact. The relationship between a meta-fact and a fact has to respect meta-modeling principle. Every layer is an instance of the higher layer. Thus, facts allow representing instances in a model. A meta-fact is defined by its name, its parameters and the meaning of each of its parameters. This definition of meta-fact can be represented as a meta meta-fact. According to this method for encoding metamodels we have developed a set of clauses represented in table 1 to which we have added additional functions used to facilitate access to different metamodel characteristics.

Meta meta-fact	MetaClass_Name(Identificateur, A1,…., An).
Meta facts	namedElement(Id,Name,Visibility). class(Id,Name, Visibility, IsAbstract). feature(Id,Name, Visibility). structuralFeature(Id,Name,Visibility, Type, Isunique, Lower,Upper,Isreadonly). property(Id,Name,Visibility, Type, Isunique, Lower, Upper, Isreadonly, Iscomposite, IsDerived).
Meta meta-fact	Association_End_Name(id1, id2).
Meta facts	ownedAttribute(IdClass, IdProperty). class(IdProperty,IdClass). superClass(IdClass, IdClass). memberEnd(Idassociation, IdProperty). association(Idassociation, IdProperty)
Meta meta-fact	General_Class(Id,A1, ...,An) : - Special_Class(Id,B1,...,Bm)
Meta facts	feature(Id,Name, Visibility) : - structuralFeature(Id,Name, Visibility, Type, Isunique, Lower, Upper, Isreadonly).

Table 1. Extract of LP clauses encoding MOF metamodels

4.1.2. Encoding metamodel changes in LP

We model differences between an original metamodel MM1 and an evolved version MM2 as a set of changes. These changes are of three kinds: Additive changes, where the evolved metamodel contains an element that was not present in the original metamodel. Subtractive changes, where the evolved metamodel misses an element which was present in the original metamodel. Updative changes, where the evolved metamodel contains an element which corresponds to an element in the original metamodel but the value of a meta-feature in MM2 is different from its value in MM1. For encoding these changes in LP formalism, we propose the following meta meta-fact to represent changes: *Change_Kind(Type, Id , [P1,…,Pn]).* Where Change_Kind can be one of three values: added, deleted or updated; Type and Id indicate respectively the type and the identifier of the element subject of the change; [P1,…,Pn] is a set of parameters according to the change kind and meta element features. This meta meta-fact corresponds to M3 level. In M2 level, we define meta-facts according to possible metamodel changes described in the literature (Wachsmuth, 2007; Gruschko, 2007; Cicchetti, 2008; Herrmannsdörfer , 2008). For instance *added (class, Idc, [Name])* represent the meta-fact which allows creating a new class *Name.*

4.1.3. Encoding evolution operators

This step permits to propose evolution operators which when applied on a metamodel version give a valid new version. It is realized only once but is the core of our proposal. In this step, two tasks are performed:

- Analyzing MOF metamodels to identify and formalize authorized atomic

evolution operators;

- Definition and formalization of composite evolution operators applicable on MOF metamodels.

Operators originate from the literature of metamodels evolution. Wachsmuth first proposes a set of operators according to the preservation of metamodel expressiveness and existing models (Wachsmuth, 2007). Gruschko et al envision a difference based approach and therefore classify all primitive changes according to their impact on existing models (Gruschko et al, 2007). Cicchetti et al list a set of composite changes used in a difference based approach (Cicchetti et al, 2007). Herrmannsdoerfer et al propose a catalog of 61operators for the coupled evolution of metamodels and models (Herrmannsdoerfer et al, 2008). These coupled operators evolve a metamodel and are able to automatically migrate existing models. In our proposal, we will construct a catalog of operators encoded in LP formalism. Furthermore operators will be organized in a way where the operator evolution and co-evolution parts will be clearly defined in order to facilitate the library extension.

Primitive operator performs an atomic metamodel evolution step that can't be further subdivided. A list of these operators considering principal evolutionary elements is given in table 2. Each operator has a number of formal parameters like class and property names. Composite operators can be decomposed into a sequence of primitive operators which have the same effect at the metamodel level but typically not at the model level. Composite or compound changes have been already considered in previous works like (Herrmannsdörfer , 2008; Cicchetti , 2008) . Therefore, in our proposal we formally define them using LP formalism, we note that further intermediate functions are represented in these rules representing condition needed to execute operations. We define only a few composite operators in this paper shown in table 3. For instance, we consider extract super class operator where a class is generalized in a hierarchy by adding a new general class and two references to their subclasses. The complete set of primitive and compound operators represents a knowledge base which will be used by the inference engine to reconstruct evolution scenarios.

Evolution operators have different impacts on models conformity (Gruschco et al, 2007). We define four categories of operators according to this impact. An operator is free impact when change performed by applying this operator is not breaking. The operator is automatic, if the change produced by applying an operator is breaking and resolvable and for this category the operator is reusable and every time migration procedure will be automatically executed. Whereas, for breaking and not resolvable changes, we distinguish between user driven operators and manual operators. Operators are user driven where migration of models cannot be completely automated and need more information from the model designer however when model migration cannot be defined automatically and adaptation solution must be provided manually by model designer the operator is considered manual.

4.2. Detection of changes

Evolution from one metamodel version to the next can be described by a sequence of changes. Two ways exist for discovering changes (Cicchetti, 2008). One way, changes applied to the metamodel have to be detected a posteriori using model comparison approaches with either generic model comparison algorithms or language-specific comparison algorithms. Another way to acquire the set of changes is to use operation recording; that is, the execution of evolution operations is tracked within the modeling environment while they are performed. The later solution can be used only if metamodels are evolved by the same tool in order to capture the evolution tracks. Our approach uses the first method.

Primitive operator definition	
create_class(Name, Visibility, IsAbstract)	make_composite(Idass, Idclass_s)
delete_class(Name)	drop_composite((Idass, Idclass_s)
create_property (Idproperty, Idclass)	make_opposite((Idass , Idclass _s, Idop, Idop_s).
create_asso(Name_a, Idclass_s, Idclass_T)	drop_opposite ((Idass , Idclass _s, Idop, Idop_s).
delete_property(Idproperty, Idclass_c)	generalize_lower(Idass, Idclass_s, L).
rename_class(Idclassc1, Name_c)	specialize_lower(Idass, Idclass_s, L).
rename-property(Idproperty, Name_p)	generalize_upper(Idass, Idclass_s, U).
make_abstract(Idclass)	specialize_upper(Idass, Idclass_s, U).
drop_abstract(Idclass)	make_identifier(Idproperty , Idclass).
add_super(Idclass_s, Idclass_G).	drop_identifier(Idproperty , Idclass).
drop_super((Idclass_s, Idclass_G)	

Table 2: Extract from primitive evolution operators in LP clauses.

Composite operator definition
Pull_up_feature(Idclass, Idproperty) :- typeof(Idproperty, property), findall(Cs, super(Idclass, Cs), Ls), delete_property(Idproperty, Ls) , create_property(Idproperty, Idclass).
Pull_up_feature(Idclass, Idproperty) :- typeof(Idproperty, association), findall(Cs, super(Idclass, Cs), Ls), getName(Idproperty, Name_a), getType(Idproperty,Type), getlower (Idproperty,L), getUpper(Idproperty,U), getComposite(Idproperty,Comp), delete_property(Idproperty, Ls) , create_asso(Name_a, Idclass, Type), specialize_lower(Idproperty, Idclass, L), generalize_upper(Idproperty, Idclass,U).
extract_super_class (Name_c, [Idclass1,..., Idclassk],[Idproperty1, …., Idpropertyj]) :- create_class(Name_c,public,true), getId (Idclass, Name_c), add_super(Idclass1, Idclass),…, add_super(Idclassk, Idclass), pull_up_featute (Idclass, Idproperty1),…, pull_up_featute (Idclass, Idpropertyj).

Table 3. Examples of composite evolution operators in LP clauses

Primitive differences between metamodels versions are classified in three basic categories: additions, deletions, and updates (i.e. modifications) of metamodel elements (Cicchetti, 2008). These differences represent atomic changes (i.e.

elementary). Detected differences are represented as elementary changes specifying fine-grained changes that can be performed in the course of metamodel evolution. There are a number of primitive metamodel changes like create element, rename element, delete element, and so on.

One or more of such primitive changes compose a specific metamodel evolution. Every difference includes essentially information about the type of change, the element subject of change and also some evolution parameters. The set of detected differences is called delta model (i.e. difference model). Delta model is translated in LP formalism by instantiating change meta-facts described in previous section. This result is used in the next phase to reconstruct the evolution scenario.

4.3. Reconstruction of evolution scenario

Reconstruction of evolution scenario is done in two steps: firstly, we reconstruct a correctly ordered evolution scenario from the delta model. In the second step, we reconstruct composite operators by means of an inference engine.

4.*3.1. Reconstructing primitive evolution operators*

In this step atomic changes calculated in the delta model and coded with facts are transformed into sequence of primitive operations called a primitive evolution scenario. We note that an operation is an instantiation of operator parameters with actual arguments. Firstly differences are converted in atomic operations. For this purpose, we use an LP inference engine with rules base in order to obtain from each atomic change in delta model the corresponding primitive metamodel evolution operator. After that, we reorder evolution operations using the priority between changes. Finally this step checks whether the evolution scenario is valid, by applying this scenario to the old metamodel version, if it results the new metamodel version then primitive evolution scenario is well ordered else the reorganization process is repeated.

4.3.2. Reconstructing composite evolution operators

In this step we reconstruct composite operation from valid primitive evolution scenario. Facts contained in the evolution scenario are passed as input to an inference engine which will use the knowledge base of operators to detect composite operations. Once the composite operation is detected, the sequence of primitive operations is merged by adding the composite operation to the base (i.e. assert fact) and consequently to the scenario and removing facts of corresponding primitive operations from the base (i.e. retract fact). This provides a new evolution scenario which will be validated by executing it on the old version of the metamodel.

4.4. Determination of migration scenarios

Migration procedures vary according to the operator category, namely free impact, automatic reusable, user driven or manual. We are interested only by the

second and the third ones. Migration procedure is encoded as a model transformation. For reusable operators, migration procedure is automatically obtained by instantiating parameters. However, for user driven operators, we have defined migration procedures by explicitly specifying some alternatives solutions to assist user. This makes proposed library different of that used in previous works. So, the library is extended with assistance options. Consequently, migration procedures corresponding to operations in the evolution scenario are sequentially composed to constitute the complete migration scenario.

4.5. Migration

This phase takes as input an instance model conforming to the initial metamodel. Migration scenario will be applied to the input model. Some parts of the scenario will be automatically executed. In other parts the system assists users in solving the changes by presenting alternative solutions specified in the migration scenario. User selects through a graphic user interface among different alternatives for each affected element, and provides any additional information required by the selected alternative. For instance, if the metamodel evolution includes the addition of a new property, solution alternatives can include the initialization of this property with a predefined value or with a derived value based on other already existing properties. Migration scenario may contain also parts where transformation must be specified manually by model designer to complete model co-evolution process.

5. Implementation and Running example

To implement a prototype of the proposed framework, we use Ecore from the Eclipse Modeling Framework (EMF) (EMF, 2015a). For the definition of rules specifying knowledge base used to infer evolution scenarios, we have adopted an adequate formalism for logic programming Prolog (Savoy, 2006). The computation of differences between metamodel versions is performed with Eclipse plug-in EMFCompare (EMF, 2015b).

As a running example, we use a Petri net metamodel (Wachsmuth, 2007). Figure 2.a and Figure 2.b show the metamodel before and after evolution as a UML class diagram MM1 and MM2 respectively. The new version MM2 invalidates existing models.

12

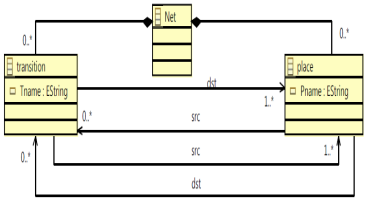

Figure 2.a. Petri net metamodel MM1

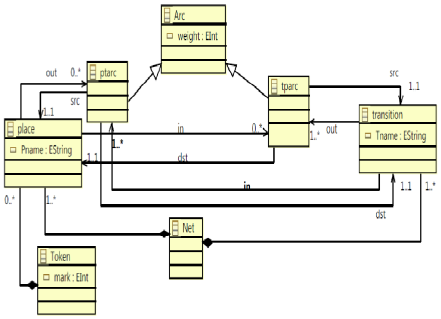

Figure 2.b. Petri net metamodel MM2

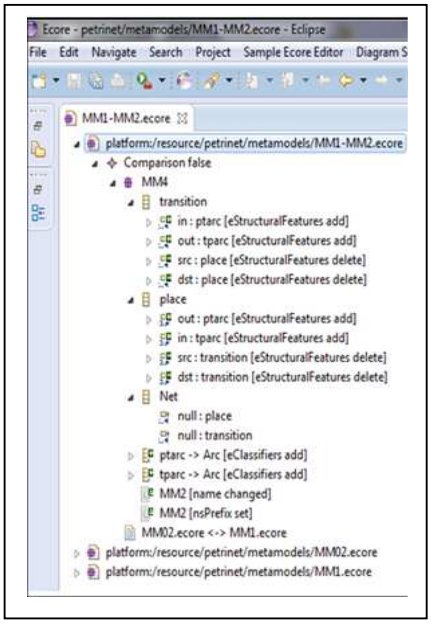

Figure 3. Delta model (MM2- MM1

```
create_class(ptarc).
create_class(arc).
create_asso(out, idc23,idc25).
add_super(idc25, idc26).
add_super(idc24, idc26).
create_property(idc26, [weight, int]).
specialize_lower(idass5,idc21,idc23,1).
specialize_lower(idass5,idc21,idc24,1)
```

Figure 4. Primitive operations

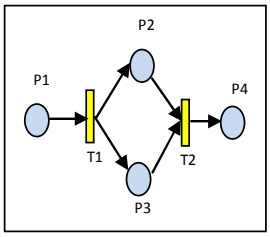

Figure 5. A sample Petri Net

First, the two versions of Petri net metamodel are provided as input. Resulted differences calculated with EMFCompare are shown in figure 3. Metamodel versions and atomic changes in the delta model are encoded in LP. the second step, we applied the inference engine to get the primitive operations from the set of changes; a part of them is depicted in figure 4.

In order to deduce composite operations we applied the inference mechanism a second time with the base of facts (BF) composed of primitive operations previously obtained and using the knowledge base defining composite operators. For instance the composite operation extract_superclass (idc26, [idc24,idc25]) will be inferred. This predicate indicates that a new class arc is created and both tparc class and ptarc class get arc as their super class, corresponding to three primitive changes. Finally, based on the validated evolution scenario, we take migration procedures defined within operators and execute them on a specific Petri net (Figure 5). To co- evolve M1, new instance objects are created according to new metaclasses and also new associations and attributes. Furthermore some associations will be dropped. During this step user will be asked about values of mark property of every token associated with place objects as well as values of weight property of different created arcs. For instance P1,P2, P3 and P4 places will be marked with values (4,2,1,0) respectively and the different arcs will be initialized with default value which is 1.

6. Conclusion

Automating the co-evolution of models and metamodels is a challenging task. In this paper we have proposed an alternative solution to this problem. Thus, our proposal illustrated a hybrid approach to guide the user in solving co-evolution issues. It takes advantages from both state-based and operator based approaches adding a new dimension to deal with this problem which is using an intelligent reasoning. This solution consists of using a library of coupled evolution operators, but unlike existing approaches, changes requiring user intervention are also integrated in the library with specific alternatives solutions to assist automatically users during migration activity. Moreover, the benefits of this approach are numerous, notably encoding of metamodels in LP clauses and the formalization of evolution operators in order to allow reasoning and to facilitate eventual extension of the operator library. The intelligent reasoning used in reconstructing evolution scenario constitutes the core of co-evolution process. It ensures a real separation between the metamodel evolution and model co-evolution and consequently between metamodel designer and model designer.

REFERENCES

Bézivin J. (2005). On the Unification Power of Models. *Software and Systems Modeling (SoSyM)*, vol. 4, n°2, p. 171–188.

Cicchetti A., Ruscio D.Di., Pierantonio A.(2007). A Metamodel Independent Representation of model Differences. *Journal of Object Technology*, special issue, vol. 6, n°9.

Cicchetti A. (2008). *Difference Representation and Conflict Management in Model-Driven Engineering*. Phd thesis.

Cicchetti A., Ciccozzi F. (2013). Towards a Novel Model Versioning Approach based on the Separation between Linguistic and Ontological Aspects. *Actes of Models and Evolution Workshop ME 2013*.

EMF (2015), Eclipse Modeling Framework, (ttp://www.eclipse.org/emf.

EMF (2015), EMFCompare, Eclipse Modeling Project, http://www.eclipse.org/emf/compare

Favre J.M. (2003). Meta-model and model co-evolution within the 3D software space. *International Workshop on Evolution of Large-scale Industrial Software Applications ELISA'03, Amsterdam*, p. 98–109.

Garcès K., Jouault F., Cointe P., Bézivin J. (2009). Managing Model Adaptation by Precise Detection of Metamodel Changes. *ECMDA-FA'09. LNCS Springer, 2009, vol. 5562, p 34-49*.

Gruschko B., Kolovos D.S., Paige R.F. (2007). Towards synchronizing models with evolving metamodels. *International Workshop on Model-Driven Software Evolution.*

Herrmannsdoerfer M., Benz S., Juergens E. (2008). Automatability of Coupled Evolution of Metamodels and Models. *Actes of Practice, In MoDELS'08. LNCS Springer*. vol. 5301, p. 645-659.

Herrmannsdoerfer M. (2010). COPE – A Workbench for the coupled evolution of metamodels and models. *Actes of SLE'10*, 2010, p. 286-295.

Herrmannsdoerfer M., Vermolen S. D., Wachsmuth G. (2011). An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. *Actes of SLE'10, LNCS, Springer, Berlin*, vol. 6563, 2011, p. 163–182.

Herrmannsdörfer M., Wachsmuth G. (2014). Coupled Evolution of Software Metamodels and Models. *Evolving Software Systems, Mens, Tom, Serebrenik Alexander, Cleve, Anthony (Eds.) Springer*, p. 33-63.

Mens T., Wermelinger M., Ducasse S. Demeyer S., Hirschfeld R., Jazayeri M.(2005). Challenges in software evolution. *Actes of 8th International Workshop on Principles of Software Evolution*, 5-6 September 2005, Lisbon, Portugal.

Narayanan A., Levendovszky T., Karsai G. Balasubramanian D. (2009). Automatic domain model migration to manage metamodel evolution. *Actes of MODELS'09, LNCS Springer*, vol. 5795, 2009, p. 706-711.

OMG (2015), Object management group, (2015). http:// www.omg.org.

OMG (2015) Meta Object Facility (MOF) Core Specification. (juin 2015). http://www.omg.org/spec/MOF/2.5

Rose L.M., Kolovos D.S., Paige R.F., Polack F.A.C. (2009). An analysis of approaches to model migration. *Actes of joint MoDSE-MCCM Workshop*, 2009.

Rose L.M., Kolovos D.S., Paige R.F., Polack F.A.C. (2010). Model migration with Epsilon Flock. *ICMT'10, LNCS Springer*,vol. 6142, p. 184-198.

Savoy, J. (2006) Introduction à la programmation logique Prolog. (2006). http://members.unine.ch/ jacques.savoy /lectures/SemCL/Prolog.pdf.

Sprinkle J., Karsai G. (2004). A domain-specific visual language for domain model evolution. *Journal of Visual Languages and Computing*. vol. 15, p. 291-307.

Wachsmuth, G. (2007). Metamodel adaptation and model co-adaptation. *Actes of ECOOP'07. LNCS Springer*, vol. 4609, p. 600-624.

MC-Sim: A Mobile Cloud Simulation Toolkit Based on CloudSim

Manel Gherari[1], Abdelkrim Amirat[2], Ridda Laouar[1] and Mourad Oussalah[3]
[1]LAMIS Laboratory, university of Tebessa, Algeria
[2]LIM Laboratory, university of Souk-Ahras, Algeria
[3]LINA Laboratory, university of Nantes, France

***Abstract*—Mobile Cloud Computing (MCC) has gained a significant attention this past years. MCC consist of migrating mobile applications from the constrained mobile devices to the cloud. This task is highly complicated and demanding, therefore several novel methods, tools and approaches have been introduced to tackle this complexity. At this point we argue that a simulation of the deployment mechanisms for accessing cloud services and testing mobile cloud applications in the cloud environment, is a mandatory phase prior to real deployment in real environment. A simulation will offer the developer a controllable, cost free environment to test and evaluate the performance of its application according to different predefined scenarios. As we can state MCC lacks in tools of simulation of its aspects; to fill this gab we propose Mobile Cloud Simulation (MC-Sim) toolkits based on CloudSim [1].**

Keywords— cloudsim; mobile cloud computing; cloudsimulation; mobile cloud simulation; mobile cloud computing modelling.

I. INTRODUCTION

Cloud Computing (CC), offers cloud services at different level (Infrastructure as service, platform as service, and software as services) to consumers that already have established a service level agreement contract based on certain negotiations with the service providers. The provisioning of cloud services is done following a payment model (pay as you go). Some well known cloud are Amazon EC2, Google App engine, Aneka, Microsoft Azure.

By migrating to the cloud, IT companies are no longer concerned by task such as setting up basic hardware and software infrastructure, so they can focus on innovation and creation of business values for their applications [2]. It should be noted that cloud based applications differs in term of composition, configuration, deployment requirements. Thus testing this kind of applications' performance, in term of provisioning policies (scheduling, allocation) is a highly demanding task due to challenges posed by the cloud like: The fact the users have heterogynous, dynamic, ever changing QoS requirements.

Using real cloud infrastructure like Amazon EC2, Microsoft azure for benchmarking application performance under variable condition is mainly constrained by the rigidity of real cloud infrastructure [1].

Motivated by this limitation, a simulated, controllable, and cost free environment is a necessity to any developer before heading to real deployment of its application in real cloud environment. To give researchers a testbed and platform to assist their research and validate their idea, several cloud simulators have been proposed this past years, like many simulation tools dedicated for cloud based applications have been proposed [1], [3], [4], [5].

Followed not by long, Mobile Cloud Computing has been introduced in order to enhance the performance of mobile device that have been invaded every personal life these days, and thus mobile user are seeking a pc like functionalities provided by Smartphone and to answer these issues, MCC outsource the processing and the storage of data from the constrained mobile devices to the elastic cloud infrastructure. As a paradigm still in its infancy MCC, unlike CC lack tools for modeling and simulation it aspects. To fill this gab we propose Mobile Cloud Simulation (MC-Sim) toolkits based on CloudSim[1].

MC-Sim is an extension of CloudSim that add more package dedicated to simulate mobile entities such as mobile device, mobile application, and more other sub-entities needed during the simulation. The main difference from CloudSim, is that MC-SIM is design in manner that separate the modeling phase from the simulation phase. First the mobile cloud application architecture is described via MC-ADL[6], based on the later a simulation entities are dynamically generated. The next step consist of executing the simulation by either defining new allocation/scheduling policies, or using the already predefined ones. MC-Sim ensure high performance of test that are open to variety and scalability since they are predefined in well specified order to be executed dynamically.

The reminder of this paper is organized as follow: Related work presenting cloud simulators will be given in section 2. Section 3 will introduce MC-SIM. Finally we conclude this paper in section 4.

II. RELATED WORK

In this section, we will summarize some cloud computing simulators.

A. *CloudSim*

CloudSim [1] is a new generalized and extensible simulation framework that is characterized by modeling and simulating of large scale infrastructure including data centers (DCs) that host several Virtual machines (Vms). Vms are allocated according to different (Space/Time) share allocation policies, with the possibility to customize the policies according to the user's requirements.

Figure 1 present the layered architecture of CloudSim.

The firs layer SimJava is a discrete event simulation engine that implements core functionalities, like queuing and processing of events, creation of systems components (Service, host, DCs, broker and Vms). Next layer is Gridsim that regroups libraries supporting modeling multiple grid infrastructure, including networks, association traffics, resources data sets, workload traces, and information services. As an extension to GridSim, CloudSim allows modeling and simulation of virtualized cloud based DCs' environment, such as dedicated management interfaces for Vms, memory, storage, and Bandwidth. As a third layer CloudSim is dedicated for the instantiation and execution of core entities (Vms, hosts, and DCs). The final layer is the user core that consists of the configuration of related functionalities hosts like number of task and the user requirements.

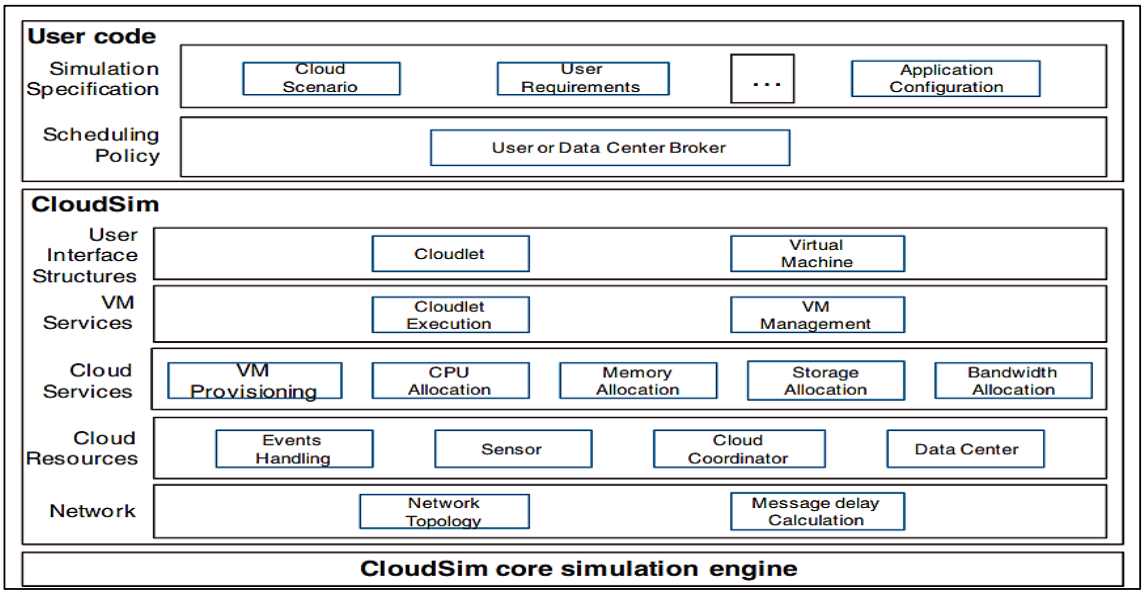

Fig. 1. CloudSim architecture

Several works have been proposed to extend and improve CloudSim, by focusing on reducing and eliminating failures that occurs during job submission [7]. Another extension of CloudSim as shown in figure 2 is Cloud Analyst [8, 9]; the toolkit proposed by Wickremasinghe, is designed to study the behavior of large scale internet applications in cloud environment while offering the possibilities to perform different simulation experiments, by simply changing parameters.

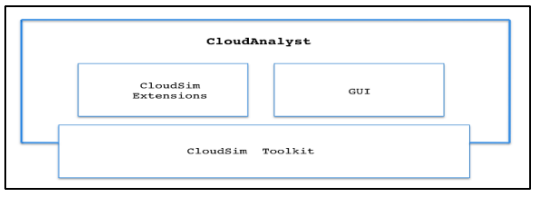

Fig. 2. CloudAnalyst Architecture.

Garg et al propose in [10] NetworkCloudSim this extension of CloudSim support modeling real time DCs and generalized application like e-commerce. The architecture of NetworkCloudSim is presented in figure 3.

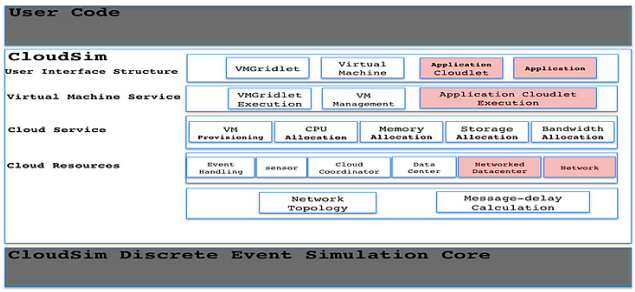

Fig. 3. NetworkCloudSim architetcure

B. SPECI

Motivated by the need to predict the performance and the behavior of future DCs, Sriram propose in [11] SPECI (Simulation Program for Elastic Cloud Infrastructure). SPECI is simulation toolkit based in SimKit [3] that allows exploration of aspect of scaling as well as performance proprieties of future DCS, by offering two main packages: one representing DCs layout and topologies, and the other encompasses the components for experiments execution and measuring.

C. iCanCloud

iCanCloud [12] is a simulator for cloud infrastructure that promises different features such as flexibility, scalability, performance and usability. iCanCloud is designed targeting the following objectives:

- Aims to conduct large scale experiments
- Provide flexible and fully customized global hypervisor for integrating any cloud broking policy
- Recreate instance type provided by cloud infrastructure
- Offer GUI interface for configuration and launching simulation.

Distinguish from other simulation toolkits, iCanCloud offer a GUI interface, also tackle the deficiency of performing a paralleled simulation across multiple machines.

As presented in figure 4, the first layer of the iCanCloud architecture is hardware layer that contains models which are in charge of modeling the *hardware* part (e.g. Disc drive, cpu, etc…). In order to guarantee a certain degree of compatibility, API is used to create application that enter the simulation with iCanCloud. Next layer is *Vm repository* that contains a collection of Vms that have been defined by users. *Cloud Hypervisor* consist of models responsible for managing all jobs instance of Vms, and cost policies. *Cloud System Module* encompasses the definition of the entire cloud system by defining hypervisor and Vms.

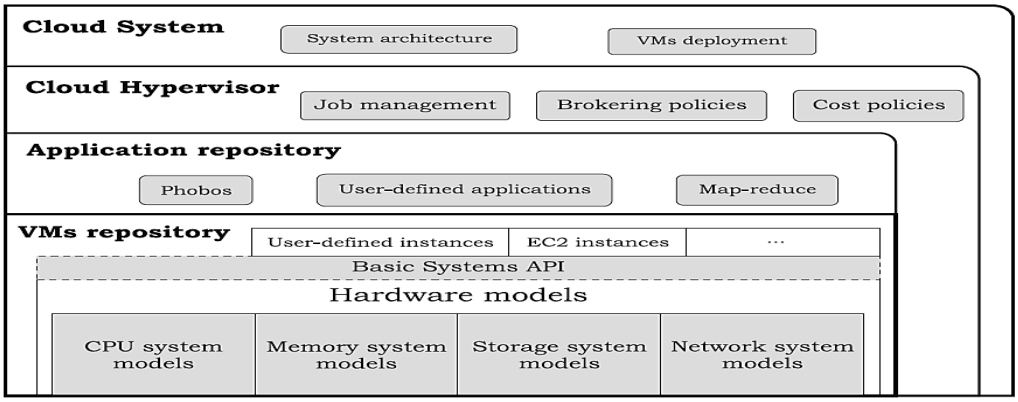

Fig. 4. iCanCloud Architecture

.

D. GreenCloud

Based on NS2 [13], GreenCloud [14, 15] is packet level cloud simulator that mainly focus on evaluating the energy cost of DCs operation. The GreenCloud architecture is presented in figure 5.

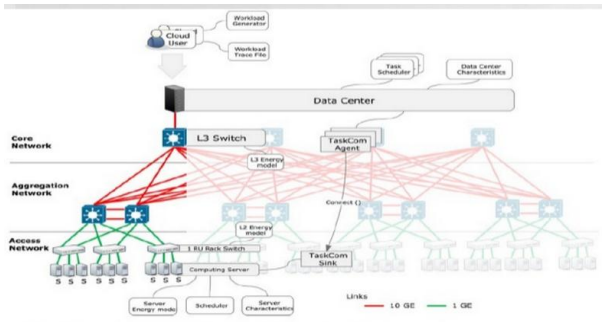

Fig.5. GreenCloud architecture

E. Open Cirrus

Established by Yhaoo and HP, Open Cirrus [16, 17] provides a testbed heterogeneous distributed DCs for systems, applications and services. Open Cisrrus aims to foster system level research in cloud computing, encouraging new cloud application research by offering a platform for real world application, and providing a collection of experimental data in order to support researcher with their high quality experiments.

Other than cloud computing , network computing also had it share of having simulation tools that are interested in network details, network protocol, and path discovery like: DaSSF [18], OMNeT++ [5, 19], OPNet [20, 21] and NS2 [13].

III. Proposed Approach

Mobile cloud computing has gained the interest of researcher this past few years, as mobile devices have invaded our lives by being a necessity and no longer a luxury. So in order to provide pc-like functionalities to mobile user, merging the cloud and the mobile computing was the important step to brighter future in mobile applications' industry.

We argued in previous work [6] that mobile cloud application need an architecture representation as any other system. In this paper and following the development cycle of mobile cloud application we believe that a simulation of the behavior of the application in cloud environment is prior and indispensable before heading to real deployment. At this phase the developer can test repeatedly its application in simulated, controllable const free environment, and ensure a pane/bug free application smoothly deployed in real mobile cloud environment.

After studying several simulations tools we have came to choose CloudSim to extend as the more adequate and highly corresponding toolkit to our vision. We mainly implemented our work in java language and the simulation in our case is based on mobile cloud architecture model described by MC-ADL [6] eclipse plug-in.

CloudSim do not offers classes to simulate mobile entities since it is not developed for mobile cloud applications simulations. Addressing this limit, we have added new simulating packages to CloudSim dedicated to mobile cloud application simulation. These later regroup classes that are defined in the same logic of CloudSim entities by abstracting the mobile cloud elements and their behaviors.

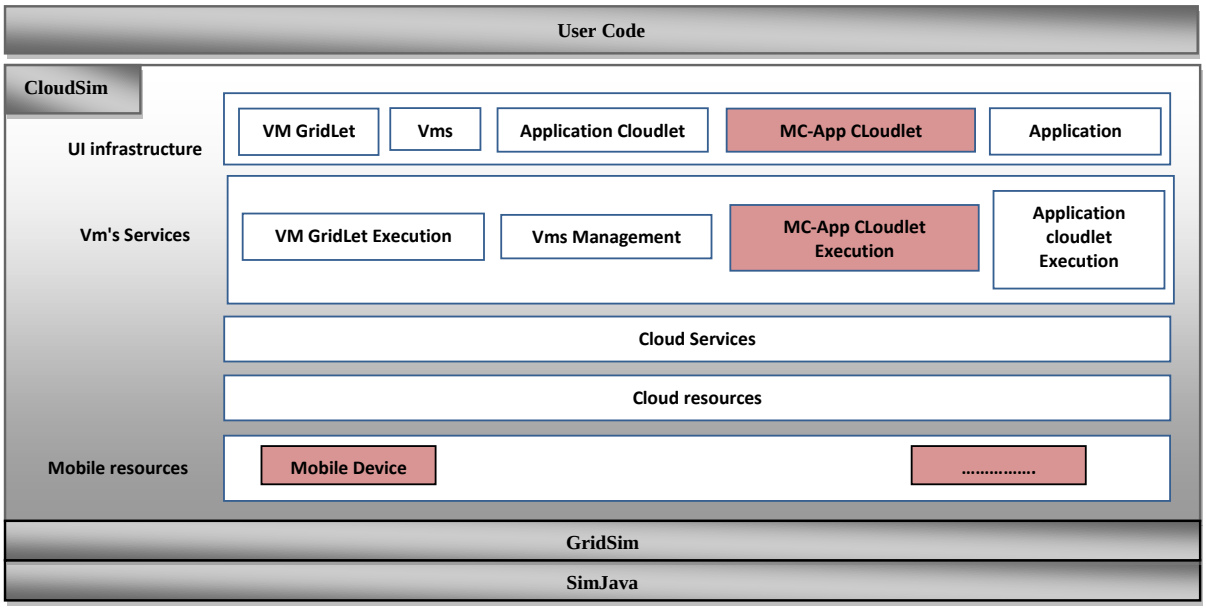

Fig. 6. MC-Sim Architecture

Figure 6 shows the MC-Sim layered Architecture based on CloudSim. *"MC-App Cloudlet"* encapsulate all the relations between the application task and the provided cloud service. "*Mobile Device"* encapsulate the information concerning the mobile device starting from its operating systems to its intrinsic and external contextual information.

It should be noted that we are interested in taking advantage of contextual information provided by the cloud and the mobile environments. This kind of information is presented in the architecture model and will be mapped later in the simulation process. Concerning the cloud, its contextual information encompasses type cloud model, type of cloud service, cloud provider, availability, Qos, payment model, and etc…

Mobile contextual information are divided in two classes:

- *Intrinsic* that regroups the permanent information as the characteristic of the mobile device (e.g. os, memory, storage capacity, processing power).
- *External* Concerning environment information like location, time, neighbor devices, and network connection.

Based on the architecture model (see figure7) representing the Mobile Cloud Application (MC-App) a *simulator* class is java program that will exploit the information in the MC-App model, more specifically its xml representation to create an extended entities corresponding the mobile and cloud elements that answer the

characteristics specification defined by the developer during the design phase. Some entities like cloud broker, allocation policies, and scheduling are created dynamically since they are predefined entities, but they can be customized later according to the developer's requirements.

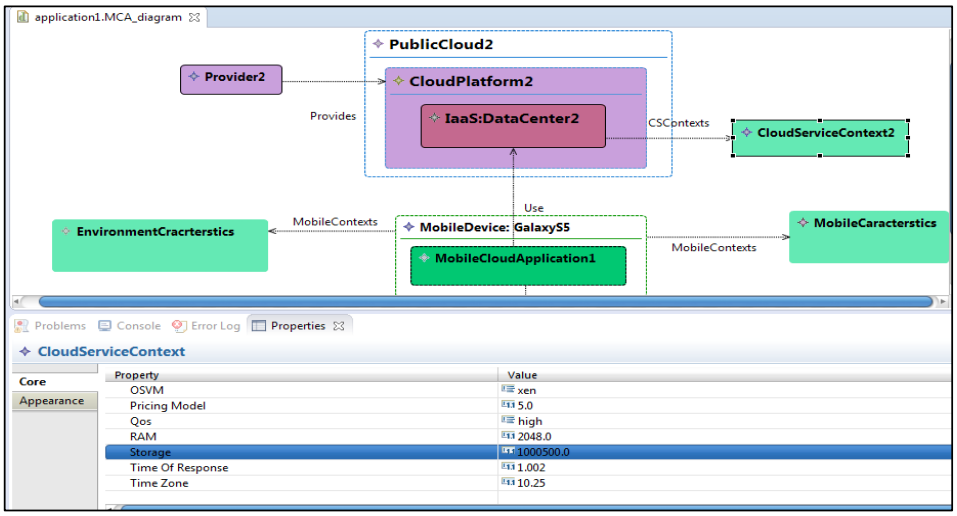

Fig. 7. Mobile cloud application architecture model.

As presented in the activity diagram (see figure 8) ; after generating the object representing the xml version of the architecture model, the later will be an input to the second activity that take it as parameter to create the two objects representing cloud entities and mobile entities. It should be noted that the developer can change the specification of the simulated entities according to its requirement.

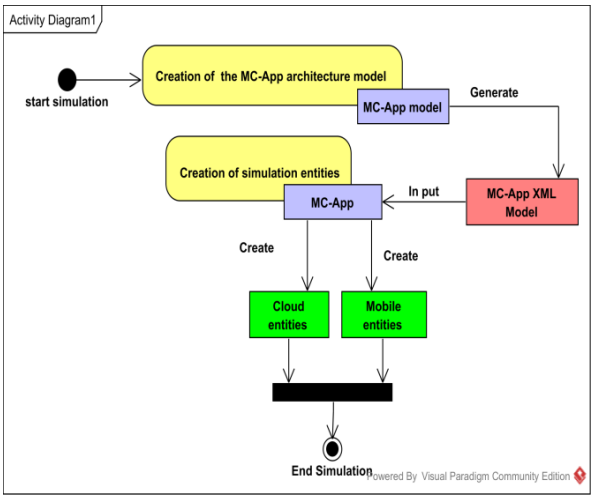

Fig. 8. Simulation execution workflow

To foster the concept of dynamic architecture evolution, we manipulate the execution of the simulation dynamically by running several simulation scenario that add or delete some cloud entities which will provoke and evolution at the architecture level.

During the simulation phase an important question must be asked is: Which mobile application task that need to be offloaded to the cloud and which one need to be executed locally ? Many researchers [22], [23], have tried to answer this question. Thus in our proposed framework we have not implemented an offloading decision algorithm, assuming that every task of the mobile cloud application will be delegated/offloaded to the cloud according to predefined or customized allocation policies in CloudSim.

IV. Conclusion

We have presented in this paper a brief stat of art of cloud computing simulation tools, highlighting the lack of simulators for mobile cloud computing. Motivated by this limit we have proposed Mobile Cloud Simulation toolkit based on CloudSim (MC-Sim). during the design of MC-Sim we have settled the following g objectives:

- Exploit the generic, independent mobile cloud architecture model described via MC-ADL[6], to dynamically create the simulation entities.
- Foster a high performance simulation process by defining several simulation scenarios that run dynamically in certain order to fulfill any test requirements.

- Simulating mobile cloud environment changes in term of adding modifying, deleting cloud resource or mobile resources to foster a certain degree of adaptability of mobile cloud application.

In this first prototype of MC-SIM, the mobile cloud application task is directly assigned to be executed by a cloud service, we plan as future work to implement a decision algorithm to decide when and how to offload/delegate certain mobile cloud application tasks to the cloud.

REFERENCE

1. Calheiros, R.N., et al., *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.* Software: Practice and Experience, 2011. **41**(1): p. 23-50.
2. Armbrust, M., et al., *A view of cloud computing.* Communications of the ACM, 2010. **53**(4): p. 50-58.
3. Buss, A. *Simkit: component based simulation modeling with Simkit*. in *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*. 2002. Winter Simulation Conference.
4. Buyya, R. and M. Murshed, *Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing.* Concurrency and computation: practice and experience, 2002. **14**(13-15): p. 1175-1220.
5. Varga, A. *The OMNeT++ discrete event simulation system*. in *Proceedings of the European simulation multiconference (ESM'2001)*. 2001. sn.
6. Gherari, M., A. Abdelkrim, and M. Oussalah, *Describing Ubiquitous Mobile Cloud Systems.* 1st international conference on Information Technology for Organization Development (IT4OD), Tebessa, Algeria, 2014.
7. Belalem, G., F.Z. Tayeb, and W. Zaoui, *Approaches to improve the resources management in the simulator CloudSim*, in *Information Computing and Applications*. 2010, Springer. p. 189-196.
8. Wickremasinghe, B., R.N. Calheiros, and R. Buyya. *Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications*. in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. 2010. IEEE.
9. Wickremasinghe, B., *Cloudanalyst: A cloudsim-based tool for modelling and analysis of large scale cloud computing environments.* MEDC project report, 2009. **22**(6): p. 433-659.
10. Garg, S.K. and R. Buyya. *Networkcloudsim: Modelling parallel applications in cloud simulations*. in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. 2011. IEEE.
11. Sriram, I., *SPECI, a simulation tool exploring cloud-scale data centres*, in *Cloud Computing*. 2009, Springer. p. 381-392.
12. Núñez, A., et al., *iCanCloud: A flexible and scalable cloud infrastructure simulator*. Journal of Grid Computing, 2012. **10**(1): p. 185-209.
13. NS2, T.N.S., *available at "http:\\www.isi.edu\nsnam\ns\.*
14. Kliazovich, D., P. Bouvry, and S.U. Khan, *GreenCloud: a packet-level simulator of energy-aware cloud computing data centers.* The Journal of Supercomputing, 2012. **62**(3): p. 1263-1283.
15. Liu, L., et al. *GreenCloud: a new architecture for green data center*. in *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*. 2009. ACM.
16. Avetisyan, A.I., et al., *Open cirrus: A global cloud computing testbed.* Computer, 2010(4): p. 35-43.
17. Campbell, R., et al. *Open cirrusTM cloud computing testbed: federated data centers for open source systems and services research*. in *Proceedings of the 2009 conference on Hot topics in cloud computing*. 2009. USENIX Association.
18. Liu, J. and D.M. Nicol, *DaSSF 3.1 user's manual.* Dartmouth College, 2001.
19. Varga, A. and R. Hornig. *An overview of the OMNeT++ simulation environment*. in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
20. Chen, M., *OPNET network simulation.* Press of Tsinghua University, 2004. **1**.
21. Chang, X. *Network simulations with OPNET*. in *Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future-Volume 1*. 1999. ACM.
22. Flores Macario, H.R. and S. Srirama. *Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning*. in *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*. 2013. ACM.
23. Kumar, K. and Y.-H. Lu, *Cloud computing for mobile users: Can offloading computation save energy?* Computer, 2010(4): p. 51-56.

Vers une approche d'adaptation du comportement énergétique d'un bâtiment intelligent

Emna Taktak [1], **Imen Abdennadher** [1],
Ismael Bouassida Rodriguez [1,2]

1. *ReDCAD Laboratory, University of Sfax, National School of Engineers of Sfax, B.P. 1173, 3038 Sfax, Tunisia*
emna.taktak@redcad.org, imen.abdennadher@redcad.org

2. *LAAS-CNRS, University of Toulouse, CNRS, Toulouse, France*
bouassida@redcad.org

RÉSUMÉ. *Plusieurs projets de recherche dont les sujets sont liés aux bâtiments intelligents et à la gestion de la consommation d'énergie ont été mené ces dernières années. Un comportement inattentif peut diminuer d'un tiers la performance énergétique d'un bâtiment. Par ailleurs, les applications de gestion de bâtiments intelligents doivent assurer un compromis entre la consommation d'énergie et le confort des occupants. Ceci est dû au fait que les espaces de vie intelligents se caractérisent par un changement fréquent du contexte. Dans ce travail, nous nous concentrons sur la conception d'un gestionnaire de bâtiment intelligent qui est capable de gérer efficacement la consommation d'énergie tout en préservant le confort des occupants.*

ABSTRACT. *Several research projects whose topics are related to Smart Buildings and energy consumption management were conducted these last years. Energy-unaware behaviour can decrease one-third from a Building's designed energy performance. Designing Smart Building applications is a challenge since these applications must ensure a trade-off between energy consumption and occupants' comfort especially in smart spaces which are characterized by a frequent context change. In this work, we focus on the design of a Smart Building Manager that is able to manage energy consumption while keeping the occupants' comfort.*

MOTS-CLÉS : *Espace de vie intelligent, gestionnaire de bâtiment intelligent, efficacité énergétique, adaptation*

KEYWORDS: *Smart spaces, Smart Building Manager, energy efficiency, adaptation*

DOI:10.3166/RIA.28.1-10 © 2014 Lavoisier

Revue d'intelligence artificielle – n° 2-3/2014, 1-10

1. Introduction

La vie moderne est assistée par une multitude de dispositifs qui sont devenus indispensables et qui consomment de plus en plus d'énergie. D'un autre côté, la consommation d'énergies non renouvelables engendre beaucoup de risques, à savoir la pollution de l'environnement, le réchauffement climatique, l'épuisement des sources d'énergie actuelles, en plus de l'impact économique. Les points que nous avons relevé motivent la gestion efficace de la consommation et l'utilisation des énergies renouvelables. Le rendement de ces énergies peut être relativement fluctuant dépendant de la zone d'installation, de la saison voir des aléas climatiques. De ce fait, il est nécessaire de gérer efficacement la consommation par rapport la production de l'énergie renouvelable pour bien en profiter. En réponse à cette problématique, de nombreux travaux de recherche ont été menés durant ces dernières années ayant comme objectif la gestion et la maîtrise de la demande énergétique. Le secteur bâtiment résidentiel et tertiaire est concerné par ces recherches puisqu'il représente un nœud énergétique important qui a un impact important sur la demande énergétique globale. Ce secteur peut profiter des contributions du domaine de la sensibilité au contexte (Khabou, Bouassida Rodriguez, 2015) ou de l'informatique ubiquitaire (Sancho *et al*., 2010).

Dans ce contexte, nous avons vu apparaître des bâtiments intelligents qui se caractérisent par la production des énergies renouvelables pour assurer une autoconsommation et faire des économies au niveau de la demande d'énergie. Des gestionnaires de bâtiments intelligents ont été développé utilisant différentes stratégies pour la réduction de la consommation globale des bâtiments. Parmi ces stratégies d'économie dans un bâtiment intelligent nous pouvons citer :

– L'arrêt des équipements inutilisés : l'utilisation des capteurs de présence qui donnent l'information à propos l'utilisation d'un tel équipement. La période d'absence pendant laquelle un équipement est jugé inutilisé varie selon l'équipement lui même et selon le type du bâtiment.

– L'adaptation dynamique des équipements selon le contexte : l'amélioration du confort dans la maison (chauffage, climatisation, ventilation, éclairage et volets/stores électriques : il s'agit de gérer les apports naturels en fonction de l'enveloppe thermique du bâtiment) [1].

– L'avertissement des utilisateurs : la disposition des conseils en temps réel aident l'utilisateur à identifier les consommations inutiles ou déplacer ses consommations vers une période plus économique.

– L'élimination des périodes de veille pour certains équipements : l'élimination de l'état en veille des appareils, et notamment des équipements multimédia, qui sont responsables d'une part notable de consommation : TV, récepteur, etc. La gestion des scénarios de veille ou de départ permet de réduire efficacement la consommation.

1. http://www.cre.fr/

– L'utilisation d'un support de stockage de l'énergie : le stockage de l'énergie peut être réalisé en intégrant les véhicules électriques à la fois comme support de stockage et comme consommateur d'électricité. La batterie du véhicule peut absorber l'énergie excédentaire en période de charge réduite et ré alimenter le bâtiment en période de charge élevée.

Malgré l'utilisation des énergies renouvelables et la mise en place de toutes ces stratégies dans un bâtiment intelligent, l'auto-consommation n'est pas toujours garantie. Il arrive que la production d'énergie renouvelable ne couvre pas les consommations courantes. Dans ce cas le basculement vers l'utilisation de l'énergie fournie par un distributeur d'énergie est nécessaire. Néanmoins, une simple adaptation effectuée sur le fonctionnement des équipements permet de diminuer la demande de l'énergie fournie par un distributeur d'énergie ou de l'éviter en conservant l'état d'auto-consommation plus de temps.

Dans ce travail, nous nous concentrons sur la conception d'une application pour un bâtiment intelligent pour améliorer sa gestion d'énergie. L'élément principal de cette application est le composant logiciel : gestionnaire de bâtiment intelligent. Ce gestionnaire est conçu pour contrôler la consommation de l'énergie. Il est capable de gérer la consommation d'énergie tout en préservant le confort des occupants. En plus de l'utilisation des stratégies d'économie, ce gestionnaire intelligent peut gérer les situations de surconsommation en exécutant une démarche d'adaptation. Notre objectif derrière l'utilisation de ce gestionnaire est d'assurer une certaine autonomie du bâtiment à travers la réduction de l'intervention humaine. Donc, le gestionnaire de bâtiment intelligent peut adapter le comportement du bâtiment selon le contexte en gérant les différents équipements du bâtiment.

Notre papier est structuré comme suit : dans la deuxième section nous présentons quelques définitions et concepts. Dans la troisième section, nous décrivons la démarche d'adaptation. Nous illustrons cette démarche à travers un cas d'étude dans la quatrième section. La cinquième section est dédiée à la conclusion.

2. Concepts et définitions

2.1. Espace de vie intelligent

Les espaces de vie intelligents sont les environnements comme les appartements, les bureaux, les musées, les hôpitaux, les campus universitaires, et les espaces extérieurs dans lesquels les objets (capteurs, dispositifs, appareils) peuvent coopérer, aussi les systèmes qui sont capables d'auto organiser en se basant sur des politiques données (Sciuto, Nacci, 2014). Cook et Das (Cook, Das, 2004) donnent une définition générique des espaces de vie intelligents : un espace de vie intelligent est capable d'acquérir et d'appliquer les connaissances sur son environnement et d'adapter à ses habitants afin d'améliorer leur expérience dans cet environnement. Les espaces de vie intelligents sont constitués de différents capteurs et d'actionneurs coordonnés par un

système central. Les capteurs rassemblent de l'information qui doit être traitée pour agir efficacement sur les actionneurs (Sciuto, Nacci, 2014).

2.2. Bâtiment intelligent

Le groupe de bâtiment intelligent européen (EIBG) définit un bâtiment intelligent comme : celui qui crée un environnement qui maximise l'efficacité des occupants du bâtiment, tout en permettant en même temps la gestion efficace des ressources avec un coût minimum du matériel et des équipements (Wigginton, 2002). Ainsi, les bâtiments intelligents sont des espaces de vie intelligents considérés comme les environnements de la prochaine génération, ils introduisent le concept de mise en réseau des équipements dans un bâtiment et le concept de l'efficacité énergétique dans un bâtiment. Dans ce travail, le bâtiment intelligent considéré est présenté dans la Figure 1. Dans ce bâtiment intelligent, le thermomètre, le panneau photovoltaïque et le compteur d'énergie intelligent sont responsables de la collecte et de l'envoi d'information au gestionnaire du bâtiment intelligent. Les actionneurs sont les lampes et les climatiseurs. L'utilisateur interagit avec le gestionnaire du bâtiment intelligent à travers l'interface utilisateur.

2.3. Adaptation

Le travail de Saha et Mukherjee (Saha, Mukherjee, 2003) définit l'adaptation comme l'opération qui consiste à apporter des modifications à un logiciel ou à un système informatique, à la fin de son développement, dans le but d'améliorer ses performances dans un contexte précis d'utilisation. En revanche, dans le cadre des systèmes sensibles au contexte, l'adaptation est étendue par la notion d'auto-adaptation. Citant ainsi la définition proposé par Oreizy et al. (Oreizy *et al.*, 1999) : un logiciel auto-adaptatif modifie son propre comportement en réponse à des changements dans son environnement de fonctionnement. Par environnement de fonctionnement, nous nous référons à tout ce qui est observable par le système logiciel, comme les données fournies par l'utilisateur, les matériels et les capteurs externes ou des mesures. Raibulet (Saha, Mukherjee, 2003) identifie trois concepts importants des systèmes adaptatifs. En premier lieu, l'adaptabilité est demandée comme réponse à des changements qui arrivent soit à l'intérieur du système, soit dans son environnement. Dans ce deuxième cas, le système est sensible au contexte. En deuxième lieu, l'adaptabilité est atteinte avec des changements que le système effectue sur lui-même. Finalement, pour considérer qu'un système est pleinement adaptatif, ces changements doivent s'effectuer lors de l'exécution du système. Ainsi, un système auto-adaptatif sensible au contexte doit respecter ces trois concepts afin de pouvoir d'auto adapter aux changements de son contexte. Nous sommes intéressés par l'adaptation de cette catégorie de systèmes intelligents à leurs changements de contexte. Comme exemple, nous nous concentrons sur les bâtiments intelligents et particulièrement sur l'entité logicielle gestionnaire de bâtiment intelligent.

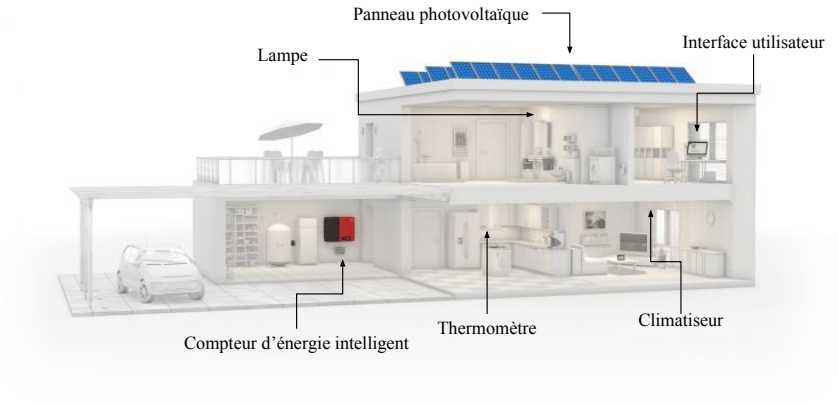

Figure 1. Le bâtiment intelligent

3. Contribution d'adaptation

Le gestionnaire du bâtiment intelligent est capable de détecter le changement de contexte et d'adapter le comportement du bâtiment en fonction de ces changements. Cette auto-adaptation nécessite une certaine intelligence au niveau du fonctionnement du gestionnaire du bâtiment afin de réduire la consommation d'énergie du bâtiment. Pour concrétiser ce but, le gestionnaire implémente des stratégies d'économie parmi celles citées précédemment ou autres. Toutefois, même avec les stratégies d'économie le gestionnaire doit contrôler la valeur de l'énergie consommée dans le bâtiment. Dans certains cas, lorsque l'énergie produite est insuffisante pour garantir le minimum d'exploitation du bâtiment, ce dernier pourrait utiliser l'énergie du distributeur d'énergie électrique public. D'autre part, le gestionnaire intelligent doit assurer le confort des occupants autant que possible.

Ainsi, lorsque le bâtiment fait face à une situation contextuelle qui conduit à une grande consommation d'énergie, l'énergie produite pourrait être incapable de répondre aux besoins du bâtiment, le gestionnaire intelligent utilise un processus d'adaptation visant la réduction de la consommation énergétique du bâtiment. Nous considérons deux types d'adaptation. Une adaptation mineure et une adaptation majeure.

– L'adaptation mineure est exécutée dans le cas où l'énergie consommée (C) dépasse l'énergie produite (P) et cet excès (C-P) atteint le seuil de l'adaptation mineure (défini par l'administrateur du bâtiment) comme illustre la Figure 2. Dans ce cas, les actions exécutées par le gestionnaire intelligent sont : augmentation de la période de mesure des capteurs et de la période d'émission des valeurs des capteurs au gestionnaire dans un premier temps. Aussi, une légère dégradation au niveau du fonctionne-

ment des dispositifs. Cette modification est légère de manière que les occupants ne vont pas sentir la dégradation.

– L'adaptation majeure est exécutée dans le cas où l'énergie consommée (C) dépasse l'énergie produite (P) et cet excès (C-P) atteint le seuil de l'adaptation majeure (défini par l'administrateur du bâtiment) comme illustre la Figure 2. Dans ce cas, les actions exécutées par le gestionnaire intelligent sont : mise en pause de certains équipements qui sont en marche (comme le lave-vaisselle ou lave-linge). Dans le bâtiment, il est possible de mettre en pause certains équipements sans altérer le service rendu au occupant (vaisselle lavée pour une certaine heure). Constatant que la surconsommation réside, le gestionnaire intelligent sélectionne l'équipement(s) qui est le plus grand consommateur dans le bâtiment et ajuste son fonctionnement avec le fonctionnement minimal permis par l'administrateur du bâtiment.

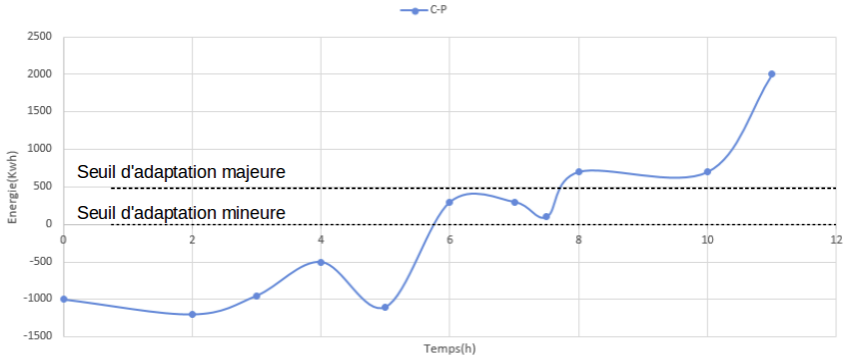

Figure 2. L'évolution de la production et de la consommation

Ainsi la démarche d'adaptation opérée par le gestionnaire peut être soit uniquement une adaptation mineure ou bien seulement une adaptation majeure ou encore une adaptation majeure suivie par une adaptation mineure comme illustre la Figure 3.

La logique de décomposition en deux types d'adaptation réside dans le fait que les actions exécutées dans le premier type d'adaptation permet de diminuer la consommation légèrement. Donc celle-ci sera exécutée lorsqu'il y a une légère surconsommation dans le bâtiment. Alors que le deuxième type d'adaptation est exécuté lorsque la surconsommation est plus importante. Cette adaptation permet de diminuer la consommation de manière importante, puisque le gestionnaire retranche la consommation de quelques équipements (ceux en pause). En plus, cette adaptation peut recourir à diminuer le fonctionnement du plus grand consommateur(s) qui permet de baisser le niveau global de consommation de manière considérable.

4. Cas d'étude : bâtiment intelligent

Le bâtiment étudié est celui présenté dans la Figure 1. Nous Considérons que ce bâtiment intelligent possède un gestionnaire de bâtiment intelligent. Ce gestionnaire

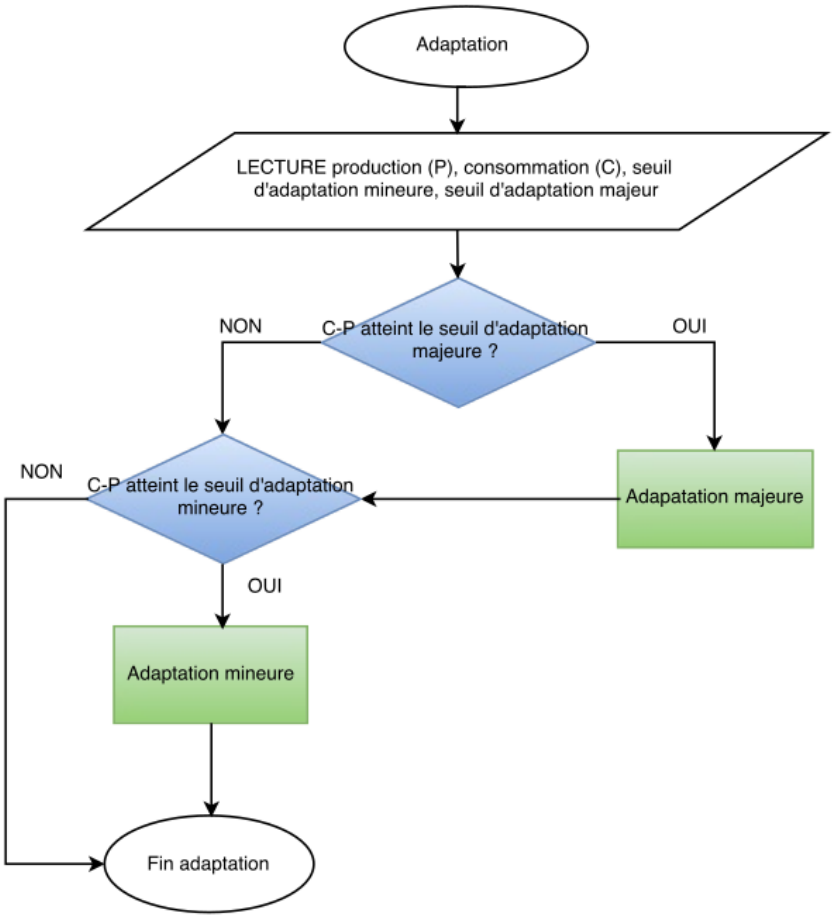

Figure 3. Démarche d'adaptation

offre plusieurs fonctionnalités utilisés par les habitants du bâtiment en utilisant une interface nommée agent utilisateur.

4.1. Composants logiciels du bâtiment intelligent

Au sein de ce bâtiment intelligent, de nombreuses entités sont interconnectées pour former un système de contrôle. Afin de permettre la coopération entre les entités du bâtiment intelligent, nous avons conçu une application qui est déployée sur les éléments matériels du bâtiment intelligent. L'application est composée de différents composants logiciels. Ces composants incluent un composant principal 'Gestionnaire du

bâtiment', un composant 'Thermomètre', un composant 'Climatiseur', un composant 'Lampe', un composant 'Compteur d'énergie intelligent', un composant 'Agent utilisateur' et un composant 'Panneau photovoltaïque'. Le compteur d'énergie intelligent calcule la consommation de l'énergie. Il est capable d'interagir avec réseau d'énergie public et d'injecter le sur plus de la production. Le thermomètre mesure la température de l'environnement régulièrement. Le fonctionnement du climatiseur et de la lampe peut être ajusté selon les besoins des utilisateurs. L'utilisateur peut accéder à l'application du bâtiment intelligent en utilisant une interface sur son dispositif mobile (smartphone, PDA, etc.). Le composant 'Agent utilisateur' est derrière cette interface. Il permet d'offrir plusieurs fonctionnalités aux utilisateurs.

Tous les composants logiciels cités sont connectés au composant 'Gestionnaire du bâtiment intelligent'. Ce gestionnaire centralise les données de l'application du bâtiment intelligent et gère le fonctionnement du bâtiment comme illustre la Figure 4. Le

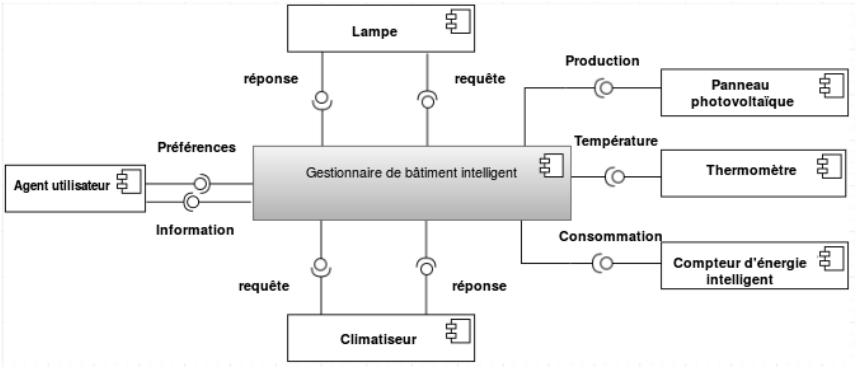

Figure 4. Composants logiciels du bâtiment intelligent

composant 'Agent utilisateur' envoie les besoins des occupants au composant 'Gestionnaire du bâtiment'. Ces besoins peuvent être une certaine valeur de l'intensité de la lampe ou aussi une valeur de la température du climatiseur. Le composant 'Thermomètre' envoie la valeur du température mesurée au gestionnaire du bâtiment périodiquement. De même le composant 'Compteur d'énergie intelligent' envoie la valeur de la consommation d'énergie. La valeur de l'énergie produite par le panneau photovoltaïque est envoyée au composant 'Gestionnaire du bâtiment' à travers le composant 'Panneau photovoltaïque'. Les composants 'Thermomètre', 'Compteur d'énergie intelligent' et 'Panneau photovoltaïque' transmettent les valeurs mesurées au gestionnaire selon des périodes d'envoi. Une période d'envoi est le temps qui sépare deux envois consécutifs au composant 'Gestionnaire du bâtiment intelligent'. Le composant 'Gestionnaire du bâtiment intelligent' peut envoyer des requêtes de consultation au composant 'Lampe' et aussi au composant 'Climatiseur'. En outre, le gestionnaire peut aussi envoyer des requêtes de mises à jour au composant 'Lampe' et aussi au composant 'Climatiseur' (pour modifier l'état de la lampe, la température du climatiseur, etc.).

4.2. Gestion de l'énergie dans le bâtiment

Le composant 'Gestionnaire du bâtiment intelligent' contrôle la valeur de l'énergie produite (P) et celle de l'énergie consommée (C) continuellement. Dans le cas où l'énergie produite par les panneaux photovoltaïques ne suffit pas pour couvrir les consommations courantes, le gestionnaire du bâtiment lance le processus d'adaptation. Considérons cet exemple de scénario d'adaptation dans le bâtiment présenté dans la Figure 1 :

1. Dans l'état initial du scénario, tous les dispositifs sont éteints, les capteurs calculent les valeurs chaque période de mesure (PM) et les envoient au gestionnaire chaque période d'envoi (PE), sachant que l'énergie produite (P) par les panneaux photovoltaïques est égale à 5000Kwh. Le seuil de l'adaptation mineure est 0Kwh. Le seuil de l'adaptation majeure est 500Kwh comme illustre la Figure 2.

2. L'utilisateur allume la lampe avec une intensité qui est égale à 1000A et le climatiseur avec une température qui est égale à 24°C. En conséquence, le compteur d'énergie intelligent envoie la valeur de l'énergie consommée (C) au gestionnaire C = 3000Kwh. C est encore inférieur à P.

3. Après une période de temps l'énergie produite diminue P = 2800Kwh. Le gestionnaire détecte que C=3000Kwh dépasse la valeur de P = 2800Kwh. Le gestionnaire procède comme suit :

a) C-P n'a pas dépassé le seuil de l'adaptation majeure, mais elle a dépassé le seuil de l'adaptation mineure. Donc, l'adaptation mineure est sélectionnée pour être appliquée par le gestionnaire.

b) Le gestionnaire augmente les périodes d'envoi des capteurs de 0,5ms (PE = ancienPE + 0,5ms) mais C>P aussi.

c) Le gestionnaire augmente les périodes de mesures des capteurs de 0,5ms (PM = ancienPM + 0,5ms) mais encore C>P.

d) Dans cette étape le gestionnaire agit légèrement sur le confort des occupants. Dans un premier temps, l'intensité de la lampe est modifiée de 1000A à (1000 - α) A. α est la valeur qui représente la sensibilité de l'œil d'un humain à la luminosité (α est définie par un expert). Aussi, la valeur de la température du climatiseur augmente légèrement de 24°C à (24 + β) C. β est la valeur de la sensibilité de l'épiderme humain à la température (β est définie par un expert). Le gestionnaire trouve ainsi que C devient inférieur à P après cette adaptation. La situation de surconsommation est arrêtée grâce à cette adaptation mineure.

5. Conslusion

Dans ce papier, nous nous sommes focalisés sur la conception d'un gestionnaire intelligent pour un bâtiment intelligent. Ce type d'environnement se caractérise par des changements fréquents du contexte auxquels le gestionnaire intelligent doit adapter le comportement du bâtiment. Parmi ces changements contextuels, le gestionnaire intelligent gère la situation de surconsommation dans un bâtiment intelligent. Nous

avons proposé une démarche d'adaptation qui permet d'éviter cette situation tout en cherchant un compromis entre la gestion de consommation et le confort des occupants. Nous avons détaillé les deux types d'adaptation qui peuvent être opérés, à savoir l'adaptation mineure et l'adaptation majeure. Nous avons présenté la conception du gestionnaire du bâtiment intelligent que nous sommes entrain d'implémenter pour améliorer l'auto-adaptation du bâtiment intelligent au changements du contexte. Comme perspectives, nous visons le déploiement réel de l'application du bâtiment intelligent pour tester la démarche d'adaptation dans des situations réelles.

Bibliographie

Cook D., Das S. (2004). *Smart environments: Technology, protocols and applications (wiley series on parallel and distributed computing)*. Wiley-Interscience.

Khabou N., Bouassida Rodriguez I. (2015). Threshold-based context analysis approach for ubiquitous systems. *Concurrency and Computation: Practice and Experience (CPE 2013)*, vol. 27, n° 6, p. 1378–1390.

Oreizy P., Gorlick M. M., Taylor R. N., Heimbigner D., Johnson G., Medvidovic N. *et al*. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, vol. 14, n° 3, p. 54–62. Consulté sur http://dx.doi.org/10.1109/5254.769885

Saha D., Mukherjee A. (2003). Pervasive computing: A paradigm for the 21st century. *Computer*, vol. 36, n° 3, p. 25–31. Consulté sur http://dx.doi.org/10.1109/MC.2003.1185214

Sancho G., Bouassida Rodriguez I., Villemur T., Tazi S. (2010). What about collaboration in ubiquitous environments. In *10th annual international conference on new technologies of distributed systems (notere'10)*, p. 143–150. Tozeur (Tunisia).

Sciuto D., Nacci A. A. (2014). On how to design smart energy-efficient buildings. In *12th IEEE international conference on embedded and ubiquitous computing, EUC 2014, milano, italy, august 26-28, 2014*, p. 205–208. Consulté sur http://dx.doi.org/10.1109/EUC.2014.37

Wigginton H. J., Michael. (2002). *Intelligent skins*. Butterworth-Heinemann.

Une approche MDE pour la modélisation et la transformation de modèle MAPS (*Mobile Agent Platform for Sun SPOTs*)

Rabah Mokhtari[1], Allaoua Chaoui[2]

1. Department of Computer Science, Faculty of Mathematics and Computer Science, University of M'sila, M'sila, Algeria
MISC Laboratory, Department of Computer Science and its Applications, Faculty of NTIC, University Constantine 2, Constantine, Algeria
ra_mokhtari2000@yahoo.fr

2. MISC Laboratory, Department of Computer Science and its Applications, Faculty of NTIC, University Constantine 2, Constantine, Algeria
allaoua.chaoui@univ-constantine2.dz

RESUME. *Le domaine des agents mobiles a tiré récemment une grande attention surtout après l'apparition des nouvelles technologies des réseaux de communication sans fil à base de dispositifs embarqués avec des ressources limitées comme les réseaux de capteurs sans fil (RCSF). Par conséquent, un ensemble de protocoles et d'outils de développement ont été réalisés pour permettre le développement et le déploiement d'applications à base d'agents mobiles sur les nœuds des réseaux d'infrastructures sans fil. Cependant, les travaux sur les approches du génie logiciel qui doivent accompagner les progressions techniques et technologiques de ce domaine sont limités et rares. Pour combler cette lacune, ce papier entre dans ce contexte et présente une approche MDE (Model Driven Architecture) pour une plate-forme spécifique dédiée aux RCSF dite MAPS (Mobile Agent Platform for Sun SPOTs). Notre approche utilise l'EMF pour définir un métamodèle décrivant les applications d'agents mobiles MAPS et implémente un outil de transformation permettant la génération automatique de code Java à partir des modèles d'applications MAPS.*

MOTS-CLES : *Agents Mobiles, MAPS, Réseaux de capteurs sans fil, MDE, EMF, ATL*

KEYWORDS: *Mobiles Agents, MAPS, Wireless Sensor Networks, MDE, EMF, ATL*

ABSTRACT. Mobile agent domain has recently been awarded with great importance especially after the emergence of new technologies of wireless communication networks based on resource-constrained devices like Wireless Sensor Networks (WSNs). Therefore, a set of protocols and plateforms has been made to develope and deploye of mobile-agent-based applications distributed on the nodes of wireless networks. However, works on software engineering methodologies that should follow the technical and technological progress in this field are again very limited. To bridge this gap, this paper falls within this context and presents an MDE approach to a specific platform designed for WSNs called MAPS (Mobile

Agent Platform for Sun SPOTs). Our approach uses EMF tool to define a metamodel describing MAPS applications and implements a transformation program allowing the generation of java code from MAPS application model

1. Introduction

Un réseau de capteur sans fil (RCSF) peut se définir comme un réseau de diapositifs infiniment petits, dits capteurs, qui sont distribués et qui travaillent ensemble spécialement pour communiquer des informations, recueillies à partir d'un champ contrôlé, via des liaisons sans fils [14]. La durée de vie d'un réseau RCSF dépend entièrement de l'energie des batteries de ses capteurs. L'énergie représente donc la ressource critique principale dans ce type de réseaux et puisque la capacité des batteries est limittée et elles sont difficiles à remplacer [15], il faut réduire le trafic des échanges entre les nœuds capteurs afin de réduire la consommation de ses energies et d'augmenter, par conséquent, la durée de vie du réseau. L. Danny B. et M. Oshima ont publié leur article célèbre intitulé : « *Seven Good Reasons for Mobile Agents »* [13] où ils insistent qu'il y aura au moins sept avantages pour commencer d'utiliser les agents mobiles dans le développement des systèmes répartis. L'un des avantages les plus importantes était « *la réduction de la charge du réseau »* du système ce qui peut maximiser la durée de vie d'un RCSF

Comme résultat, certaines plates-formes ont été développées à base du paradigme d'agent mobile pour objectif d'implémenter des applications RCSF. F. Chien-Liang et al ont présenté **Agilla** [6] une plate-forme permettant l'injection des agents mobiles dans un RCSF à base du système d'exploitation TinyOS. **ActorNet** est une autre plate-forme TinyOS conçue spécialement pour les nœuds d'un RCSF de type "Mica2" [16]. Un *framework* a été publié sous le nom **AFME** (*Agent Framework Micro Edition*) qui est une plate-forme légère basée sur la technologie J2ME et l'outil *Agent Factory* [8] et utilise le langage AFAPL (*Agent Factory Agent Programming Language*). Le service de migration de la plate-forme AFME utilise les deux protocoles "radiogram" et "radiostream" de Sun SPOT [1].

Avec cette rénovation technologique de plates-formes de réseaux sans fil à base d'agents mobiles, on ne trouve qu'un peu de travaux de génie logiciel accompagnant cette progression. Nous cherchons dans ce papier à combler l'écart entre les méthodologies des systèmes multi-agents et les applications des RCSF à base d'agents mobiles. Nous avons choisi de travailler sur une plate-forme spécifique dite **MAPS** (*Mobile Agent Plateform for Sun Spot*) : une plate-forme d'agents mobiles à base de la technologie Sun SPOT (*Sun Small Programmable Object Technology*) et du langage Java [2]. Nous suivons une approche MDE (*Model Driven Architecture*) qui utilise l'outil EMF (*Eclipse Modelling Framework*) [9] pour définir un métamodèle pour les applications d'agents mobiles MAPS et un outil de transformation ATL (*Atlas Transformation Language*) [4] pour automatiser la génération de codes Java équivalents. Pour montrer l'utilité de notre approche, ce papier présente une étude de cas couvrant presque tous les aspects introduits par cette plate-forme d'agents mobiles.

Le reste de ce papier est organisé comme suit. La section 2 présente un background pour les deux standards EMF et ATL. La section 3 rappelle l'architecture de la plate-forme d'agents mobiles MAPS. La section 4 décrit notre contribution ainsi qu'un métamodèle dédié aux applications d'agents mobiles MAPS. Pour montrer l'utilité de notre approche, la section 5 présente une étude de cas de modélisation et de génération automatique de code. La section 6 est consacrée aux travaux en relation. La dernière section présente une conclusion générale et des perspectives.

2. Background (EMF & ATL)

L'EMF [9] est un outil Eclipse permettant la construction de nouveaux outils à partir de la définition des métamodèles de type « Ecore ». L'EMF unifie les deux standards XMI (*XML Metadata Interchange*) et UML (*Unified Modeling Language*) ainsi que le langage java pour le développement de nouvelles applications. Grâce à l'EMF, plusieurs autres outils et standards, comme le langage ATL : *Atlas Transformation Language* [4], sont apparus pour permettre la manipulation et la transformation de modèles XMI créés à travers les métamodèles construits par le développeur.

3. MAPS (Mobile Agent Platform for Sun SPOTs)

MAPS [2] est un framework innovant dédié aux RCSF basé sur le langage java et la technologie Sun SPOT qui permet la programmation orienté-agent des applications RCSF. L'architecture MAPS consiste en un ensemble de composants qui peuvent s'interagir via l'échange d'événements. Chaque composant offre un ensemble minimal de services aux agents mobiles. Le comportement dynamique d'un agent mobile MAPS est modélisé par un automate d'états finis multi-plan (voir la section 3.1). Les services offerts incluent, en particulier, la transmission des messages, la création des agents, le clonage, la migration, la synchronisation et les accès faciles aux ressources des nœuds capteurs [3].

3.1 L'architecture d'un agent MAPS

MAPS est une plate-forme d'agents mobiles de poids léger (ou *lightweight* en anglais) conçue à base de la technologie Sun SPOT. Elle offre un framework, basé sur le langage java et sa machine virtuelle J2ME, pour le développement d'applications RCSF. La figure 1 montre l'architecture interne d'un agent mobile MAPS. Cette architecture consiste en ***(i)*** un ED (*Event Dispatcher*) : un composant permettant à l'agent d'émettre un événement aux plans responsable de son traitement ; ***(ii)*** MPSM (*Multi-plane State Machine*) : le comportement dynamique d'un agent peut être modélisé par un Automate d'Etats Finis Multi-Plan (ou *MPSM*) ; ***(iii)*** LV (*Local Variables*) : le composant LV représente les variables de données définies localement dans un plan particulier ; ***(iv)*** LF (*Local Functions*) : le composant LF représente les fonctions déclarées localement dans un plan peuvant

accéder aux variables locales (LV) ; ***(v)*** GV (*Global Variables*) : le composant GV représente les variables globales d'un agent par rapport à ses plans ; ***(vi)*** ECAA (*ECA-Based Automata*) : le composant ECAA modélise le comportement dynamique d'un agent mobile dans un plan particulier. Il consiste en un ensemble d'états finis et un ensemble de transitions, en exclusion mutuelle, reliant ces états. Une transition est étiquetée par des règles *ECA* qu'on peut exprimer par (E[C]/A) où les caractères E, C et A représentent respectivement Evénement, Condition et Action. Le déclenchement d'une transition se fait selon le mécanisme suivant : supposant que l'agent mobile, de la figure 1, se trouve à l'état ***s1*** et que la transition ***t1*** est étiquetée par (***e1*** [***c1***] / ***a1***). La transition ***t1*** ne peut être déclenchée que lorsque l'événement ***e1*** soit apparu et la condition ***c1*** soit évaluée à la valeur logique ***vraie*** ; dans ce cas l'agent doit exécuter l'action ***a1*** et passe à l'état ***s2***.

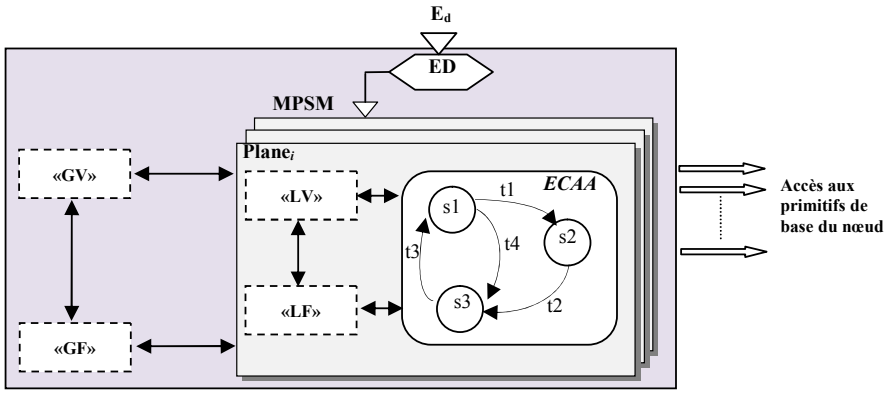

Figure 1- L'architecture d'un agent mobile MAPS [3].

3.2 Migration des agents MAPS

Le mécanisme de migration des agents de la plate-forme MAPS consiste en trois composants : un serveur MAEE (*Mobile Agent Execution Engine*) source, l'agent demandeur d'une migration et un serveur MAEE destination. Pour se déplacer à un autre nœud capteur, l'agent mobile sollicite son serveur MAEE via un message de type "*askForMigration(address)*" [3].

4 Métamodélisation et transformation de modèles d'agents mobiles MAPS

Nous avons parlé, dans la section 3, de la plate-forme d'agent mobile MAPS. Notre objectif est de fournir une approche MDE pour la modélisation et la transformation de modèle de ses applications. Nous insistons dans cette section sur l'outil EMF et le langage de Transformation ATL.

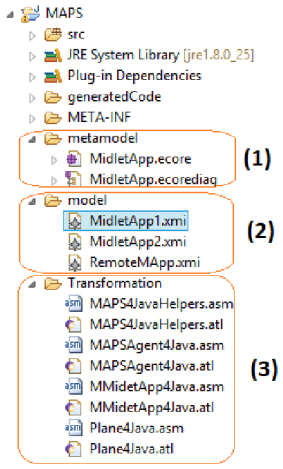

Figure 2-. Projet EMF pour MAPS

La partie **(1)** de notre projet EMF, montrée dans la figure 2, contient notre métamodèle dédié aux applications MAPS détaillé dans la section 4.1 La partie **(2)** montre des exemples de modèles créés à partir de ce métamodèle. Tandis que la partie **(3)** comprend les unités du programme de transformation ATL (voir la section 4.2).

4.1 Un métamodèle pour les applications d'agents mobiles MAPS

Notre métamodèle montré dans la figure 3 réunit l'ensemble de concepts les plus importants nécessaires à la modélisation d'une application MAPS. Ce métamodèle est décrit grâce à un diagramme de classes UML. Chaque classe représente en réalité un modèle abstrait aidant à la conception d'une application d'agents mobiles MAPS. Le point fort de notre métamodèle est sa puissance de modéliser les plans d'un agent mobile MAPSAgent. Par conséquent, il peut modéliser le comportement dynamique de ce type d'agent ce qui facilite, par la suite, la génération automatique de la totalité du code Java correspondant sauf des parties bien déterminées, par des commentaires générés, que le développeur doit compléter pour développer son application MAPS.

La partie la plus compliquée dans notre métamodèle est celle qui modélise les plans des agents et plus exactement qui dépend des conditions de déclenchement des transitions. Après une étude approfondie de l'architecture des automates modélisant le comportement dynamique de l'agent mobile de la plate-forme MAPS présentés par Aiello et all dans [2] et [3] on a déduit qu'une condition dépend soit de l'occurrence d'un événement et/ou d'un ou de plusieurs variables locales et globales.

4.2 Transformation ATL de modèle d'applications d'agents mobiles MAPS

Pour implémenter notre transformation, nous avons défini trois requêtes (*query*) et une seule bibliothèque (*library*). Par exemple, la requête « plane4Java », illustrée dans la figure 4, parcourt tous les modèles de type « Plane » afin de générer ses classes java correspondantes.

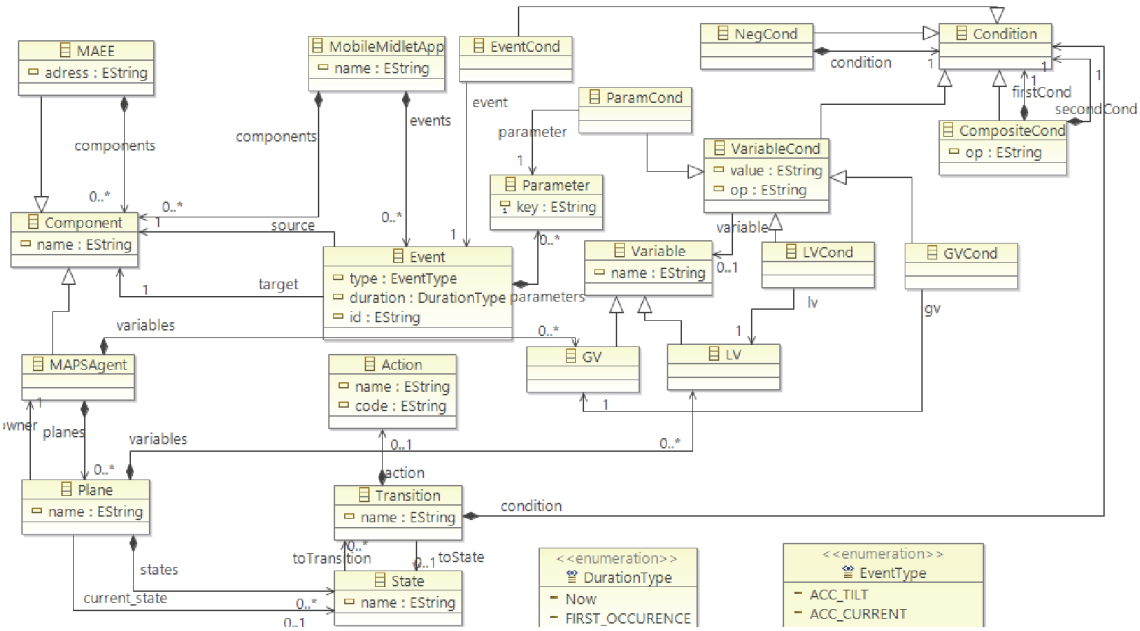

Figure 3- Métamodèle des applications d'agents mobiles MAPS.

MAPS4JavaHelpers.atl | MMidetApp4Java.atl | MAPSAgent4Java.atl | Plane4Java.atl

```
query plane4Java = midletApp!Plane.allInstances()->
                   select(plane | plane.oclIsTypeOf(midletApp!Plane))->
                   collect(plane| plane.toJavaClass().writeTo(
                   'D:/Eclipse WorkSpaces/workspace_indigo/MAPS/generatedCode/'+
                   plane.name.substring(1,1).toUpperCase()
                   +plane.name.substring(2,plane.name.size())
                   +'_'+plane.owner.name+'.java'));
uses MAPS4JavaHelpers;
```

Figure 4- Transformation ATL.

5 Etude de cas (Une application de contrôle à distance à base d'agents MAPS)

Pour montrer l'efficacité de notre approche nous avons utilisé l'étude de cas présentée dans [2]. Cet exemple contient presque toutes les variantes de modèles qu'on peut trouver dans une application MAPS. Cet exemple représente une application de contrôle à distance utilisant deux nœuds capteurs et consiste en trois agents interactifs : *DataCollectorAgent*, un agent sert à collecter des données relatives aux nœuds capteurs Sun SPOT (accéléromètre, température, clarté) ; *DataMessengerAgent* : un agent sert à communiquer les données collectées du nœud de sensation à une station de base ; *DataViewerAgent* : permet d'afficher les données collectées et reçues.

5.1 Exemple (Modélisation des plans des agents DataCollectorAgent et DataMessengerAgent)

Les deux diagrammes d'automate d'état finis montrés dans les figures 5 et 6 modélisent respectivement les plans du comportement des deux agents *DataCollectorAgent* et *DataMessengerAgent*. Le comportement de ces agents est modélisé à travers les règles de type (E[C]/A) détaillées dans la section 3.1.

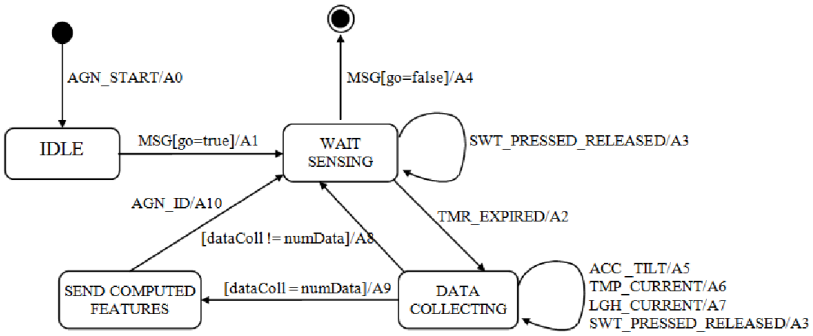

Figure 5- Modèle du plan de l'agent MAPS DataCollectorAgent

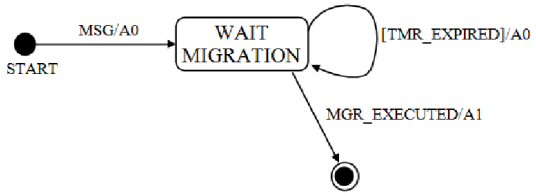

Figure 6- Modèle du plan de l'agent MAPS DataMessengerAgent

5.3 Transformation du modèle MAPS de l'application de contrôle à distance

Nous avons présenté dans les deux sections précédentes 5.1 la modélisation des plans des comportements des deux agents *DataCollectorAgent* et *DataMessengerAgent*. Ces deux types d'agent présentent deux bons exemples pour comprendre comment modéliser une application RCSF de la plate-forme MAPS. La figure 7 montre une vue globale du modèle XMI de l'application de contrôle à distance créé grâce à notre métamodèle présenté dans la section 4.1. Nous voyons, que la partie la plus compliquée dans un modèle d'agent MAPS est celle qui dépend de ses plans. Cependant, du fait que notre outil permet de créer des modèles pouvant décrire le plan d'un agent de cette plate-forme, on peut donc documenter le comportement de cet agent dans un format d'échange de modèle standard XMI (voir les deux figures 8 et 9).

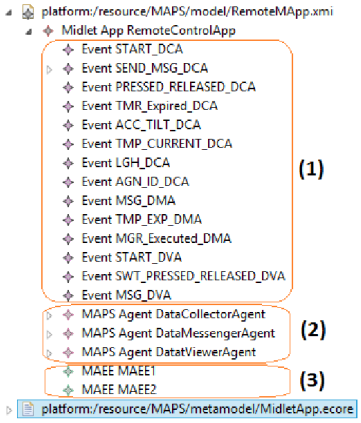

Figure 7- Un modèle XMI créé pour l'application de contrôle à distance à base d'agents MAPS.

La figure 7 contient trois parties principales : **(1)** l'ensemble d'événements traités par les différents agents de l'application modélisée. **(2)** les trois agents MAPS (DataCollectorAgent, DataMessengerAgent et DataViewerAgent). **(3)** deux serveurs représentant les deux nœuds capteurs de l'application. Les deux figures 8 et 9 montrent deux parties XMI modélisant les deux agents DataCollectorAgent et DataMessengerAgent.

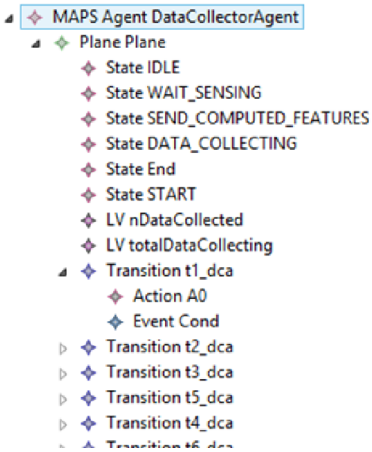

Figure 8- Modèle XMI de l'agent « DataCollectorAgent »

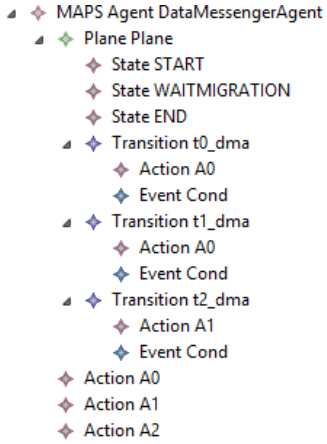

Figure 9- Modèle XMI de l'agent « DataMessengerAgent »

5.4 Exemple de génération de code

Grâce à la transformation ATL, définie dans la section 4.2, on a généré à partir du modèle XMI de l'exemple décrit dans la section 5.3, l'ensemble de classes java montré dans la figure 10. La figure 11, par exemple, montre le code généré de la classe « DataCollectorAgent.java ». Cette classe utilise à la ligne 5 une autre classe *Plane_DataCollectorAgent* qui représente en vérité le plan modélisant le comportement dynamique de cet agent.

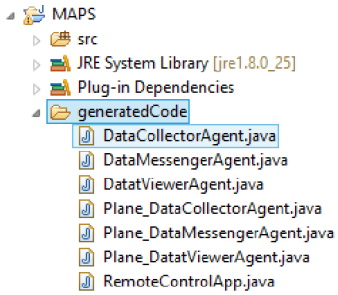

Figure 10- Classes Java générées pour l'application de contrôle à distance à base d'agents MAPS

```
1  //generated code for DataCollectorAgent Mobile Agent
2  public class DataCollectorAgent extends Agent{
3  public DataCollectorAgent(String id,
4          String executionAgentAddress) throws
5          NoSuchMailboxException {
6      super(id, executionAgentAddress);
7      Plane_DataCollectorAgent plane= new
8              Plane_DataCollectorAgent(this);
9      multiplaneStateMachine.addElement(tap);}
10 }
```

Figure 11- Classe Java générée à partir du modèle de l'agent DataCollectorAgent

La figure 12 montre une partie de la classe java générée pour le plan *Plane_ DataMessengerAgent*. Nous voyons qu'il ne reste au développeur qu'ajouter les codes des actions aux endroits des commentaires générés au dedans du code de la classe java.

```
1  //generated code for Plane_DataMessengerAgent Plane
2  public class Plane_DataMessengerAgent extends Plane {
3  private static final byte WAITMIGRATION=2, END=3 , START=1 ;
4  Public HelloWorldPlane(Agent agent) {
5      super(agent);
6      this.currentState = START;
7          this.agent.startAgent();
8      }
9  public void eventHandler(Event event) {
10     switch(this.currentState) {
11     if(event.getName()==Event.TMR_EXPIRED){
12     this.currentState = WAITMIGRATION;
13     //you should complete the code of the Action "A0" here
14     }
15     if(event.getName()==Event.MGR_EXECUTED){
16     this.currentState = END;
17     //you should complete the code of the Action "A1" here
18     }
19     if(event.getName()==Event.MSG){
20     this.currentState = WAITMIGRATION;
21     //you should complete the code of the Action "A0" here
22     }
23     }
24 }
25 }
```

Figure 12- Une partie de la classe java générée « *Plane_DataMessengerAgent* »

6. Travaux en relation

Avec la multitude de méthodologies d'ingénierie des applications d'agents mobiles, nous notons que ces travaux sont rares et très limités dans le domaine des applications de réseaux de capteurs sans fil (RCSF). A. Di. Marco et al [10] ont proposé une approche MDE permettant de modéliser et de générer automatiquement

des codes d'agents mobiles Agilla [7] (une plate-forme RCSF à base d'agents mobiles). Cette approche considère UML comme langage de modélisation et consiste en un profil UML des modèles d'agents mobiles Agilla et une transformation M2C (*Model-to-Code)*. Berardinelli et al [5] ont proposé pour Agilla une approche MDE basée sur UML et un outil de simulation dit *Cameo Simulation Toolkit*©. Cette approche consiste en la modélisation UML d'une application RCSF grâce à l'AMF (*Agilla Modeling Framework*). Les agents sont modélisés par des diagrammes d'activité UML. Pour la génération de code, cette approche prend en entrée un modèle UML d'un agent mobile puis le transforme en code Agilla correspondant. Basant sur une méthode formelle, H. Du et D. Peterson [11] ont utilisé les Réseaux De Petri Stochastique Généralisé (RDPSG) pour la modélisation et la simulation du comportement et la migration des agents mobiles dans un RCSF composé d'un serveur et d'un ensemble de nœuds et de ressources. Ce travail se focalise essentiellement sur la proposition d'un sélectionneur aléatoire transition appelé R3 (*Rolling Rocks Random*) pour briser les conflits entre les transitions immédiates afin de maximiser l'efficacité du RCSF et de bien balancer l'attente au niveau de la queue du serveur.

7. Conclusion et perspectives

Ce travail s'inscrit dans le contexte de l'ingénierie des applications d'agents mobiles dans les systèmes de réseaux de capteurs sans fil (RCSF) et plus particulièrement d'une plate-forme spécifique dite MAPS *(Mobile Agent Platform for Sun SPOTs)*. Nous avons suivi, dans ce travail, une approche MDE (*Model Driven Architecture*) qui présente un métamodèle décrivant l'ensemble de concepts nécessaires à la modélisation d'une application MAPS et un outil de transformation et de génération de code java. Un modèle MAPS définit, pour une application d'agents mobiles, tous ses composants et ses différents agents mobiles ainsi que l'ensemble des événements décrivant la communication entre eux. Une partie importante de notre métamodèle permet de décrire les deux aspects structurels et comportementaux de l'ensemble des agents (mobiles et stationnaires) appartenant à une même application ce qui facilite, par la suite, la génération d'une partie importante de code java nécessaires à l'implémentation finale de l'application. Pour illustrer l'utilité de notre approche, nous avons présenté un exemple d'étude de cas comprenant les parties les plus importantes qu'une application MAPS peut avoir. Comme perspectives, nous envisageons d'implémenter un formalisme correspondant à notre métamodèle MAPS dans un cadre formel, comme la méthode B [12], afin de valider mathématiquement nos modèles à travers une démarche de preuve de théorèmes.

Bibliographie

[1] Aiello, F., Bellifemine, F., Fortino, G., Galzarano, S., & Gravina, R. (2011). An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks. Engineering Applications of Artificial Intelligence, Elsevier , 1147–1161.

[2] Aiello, F., Carbone, A., Fortino, G., & Galzarano, S. (2010). Java-Based Mobile Agent Platforms for Wirless Sensor Networks. proceedings of the International Multiconference on Computer Science and Information Technology, IEEE , 165-172.

[3] Aiello, F., Fortino, G., Gravina, R., & Guerrieri, A. (2009). MAPS: A Mobile Agent Platform for Java Sun SPOT. proceedings of the 3rd International Workshop on Agent Technology for Sensor Networks (ATSN-09), jointly held with the 8th International Joint Conference on Automonous Agent and Multi-agent Systems (AAMAS-09), 12th May, Budapest, Hungary .

[4] ATL. Atlas Transformation Language. Consulté le 8 5, 2015, sur https://eclipse.org/atl/

[5] Berardinelli, L., Di Marco, A., Pace, S., Marchesani, S., & Pomante, L. (2013). Modeling and Timing Simulation of Agilla Agents for WSN Applications in Executable UML. Computer Performance Engineering. Volume 8168 of the series Lecture Notes in Computer Science. , pp. 300-311.

[6] Chien-Liang, F., Gruia-Catalin, R., & Chenyang, L. (2005). Mobile Agent Middleware for Sensor Networks: An Application Case Study. IEEE , 382-387.

[7] Chien-Liang, F., Gruia-Catalin, R., & Chenyang, L. (2005). Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05) , 653 - 662.

[8] Conor, M., Gregory M.P., O., Michael J., O., & Richard, T. (2008). Agent Migration and Communication in WSNs. IEEE , 425-430.

[9] Dave, S., Frank, B., Marcelo, P., & Ed, M. (2008). EMF: Eclipse Modeling Framework,Second Edition. Addison-Wesley.

[10] Di Marco, A., & Pace, S. (1-5 July 2013). Model-Driven Approach to Agilla Agent Generation. Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International , pp. 1482-1487.

[11] Du, H., Qi, H., & D. Peterson, G. (2003). Modeling Mobile- Agent-based Collaborative Processing in Sensor Networks Using Generalized Stochastic Petri Nets. IEEE International Conference on Systems, Man and Cybernetics, 2003. Volume 1. , pp. 563-538.

[12] J.R., A. (1996). The B-book: Assigning programs to meanings. Cambridge University Press.

[13] Lange, D. B., & Oshima, M. (March 1999). Seven Good Reasons for Mobile Agents. COMMUNICATIONS OF THE ACM , 88-89.

[14] Priyanka, R., Kamal Deep, S., Hakima, C., & Jean, M. B. (9 October 2013). Wireless sensor networks: a survey on recent developments and potential synergies. Springer , 1-48.

[15] Rais, A., Kada, A., Bouragba, K., Ziyati, E., & Ouzzif, M. (2015). Knowledge Discovery in WSN Using Mobile Agents. IEEE Conference Publications , 1-6.

[16] Young Min, K., Sameer, S., Kirill, M., & Agha, G. (2006). ActorNet: An Actor Platform for Wireless Sensor Networks. AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. ACM , 1297–1300.

Towards a classification for software architecture evolution studies

Noureddine Gasmallah[1,2,3],Mourad Oussalah[2],Abdelkrim Amirat[3]

1. University of Annaba
Sidi Amar, 23000 Annaba, Algeria
gasmallahedi@yahoo.fr

2. LINA, University of Nantes
2 Rue de la Houssinière BP 92208, 44322Nantes Cedex 03, France
mourad.oussalah@univ-nantes.fr

3. University of Souk-Ahras
RN 16 Route d'Annaba BP 1355, 41000 Souk-Ahras, Algérie
abdelkrim.amirat@yahoo.com

ABSTRACT. Software evolution is a logical result of the improvement efforts that stakeholders deem necessary to meet and comply with new requirements. The motivation and unprecedented momentum that fueled such interest in software evolution is the capability for software architects to make anticipations on possible changes during modeling. Consequently, software architecture as part of whole must evolve to preserve the consistency of the software at the higher levels. In return, several efforts have been made to handle evolution at the architectural levels. Nevertheless, very substantial efforts still need to be made from the architectural viewpoint within software architecture evolution. Hence, the need to frame these studies by providing a common vocabulary is appropriate to determine the field coverage. In this paper, we propose a conceptual framework for classifying software architectural evolution studies within the field. By considering four dimensions, this framework assesses, compares and helps to provide knowledge about the main direction of a given evolution approach. The proposed taxonomy reveals six classes of architectural evolution.

KEYWORDS: SOFTWARE ARCHITECTURE, MODEL OF EVOLUTION, TAXONOMY, CLASSIFICATION

DOI:10.3199/JESA.45.1-n © Lavoisier 2012

Nom entier de la revue – n° 1/2012, 1-5

1. Introduction

Software Evolution has become a major concern for stakeholders involved in the process of designing and modeling computer systems. Because of the rudimentary nature of software to evolve and change over time to account for new requirements and different needs, software evolution is considered as a vital pillar within the area of software engineering to ensure consistency and maintainability as well as to allow the system to open up for new directions and strategic opportunities. Further, even the software architecture must evolve within existing software systems (cf Jazayeri, 2005). Software architecture constitutes the favorable blueprint for supporting such evolvability at higher modeling levels. This implies that evolution could be handled at the earlier modeling phases where changes could be anticipated. It is recognized that over the last two decades, numerous studies have been developed to express the effort-making made by researchers in the field. In return, few studies have been devoted to present taxonomies for software architecture evolution. Even those that are significant, are either specialized in a software architecture evolution as knowledge re-usability (cf Ahmad et al., 2014) or addressing software evolution from an overall view perspective (cf Williams et al., 2010; Buckley et al., 2005). Buckley et al. (2005) proposed a taxonomy of software change based on characterizing the mechanisms of change and the factors that influence these mechanisms. Whilst, Williams et al. (2010) suggested that the key solution to address software changes is to characterize their causes and effects which can be used to illustrate the potential impact of the change. Breivold et al. (2012) worked on the classification in which authors provide a systematic review of architecting for software evolvability. This study does not expose any explicit framework able to position a given approach. Because of such dearth when it comes to the classification of architectural evolution methods based on a well-defined taxonomy in tandem with the fact that there is no other study which attempts to introduce a conceptual framework for evolution wherein a set of dimensions are mainly based on architectural aspect of software evolution. In this paper, we provide a conceptual framework based on the major concerns of evolution. From an architectural viewpoint, the technical answers to 5Ws questions (What, Where, How, Who and When) will highlight the main dimensions affecting software architecture evolution considering an architectural viewpoint. This study aims to show, through an application example of two well-known architectural evolution approaches, how approaches are: first, positioned and classified to provide an outline of the coverage offered using the defined evolution conceptual framework, secondly compared to select the relevant studies related to a certain situation. Our motivation is driven by the need to identify certain specifications according to which approaches can be classified. The remainder of this paper is structured as follows: during the second section, we introduce the main dimensions of the proposed conceptual framework. The third section exhibits our taxonomy for software evolution. In the fourth section, all evolution dimension facets are modeled using a meta-model for evolution. A conclusion is drawn within the last section.

2. Conceptual framework for architecture evolution

A framework is a reusable design and implementation used to evaluate and to improve certain field practices (cf Johnson, 1992). However, the conceptual framework consists to provide answers about what, where, who and how software architecture evolves. Consequently, the conceptual framework is based on four dimensions.

2.1. Hierarchical levels dimension (Who?)

Modeling levels (M_0 to M_2). Software systems must be mapped over several levels, from lower-level (codes) constructs to higher-level constructs, to try to ensure convergence (cf Shaw et al., 1995). In the present work, modeling level refers to one of the four levels (M_0 to M_3) as defined by the OMG (cf Bézivin, 2005). Thus, evolution can be performed in one or more of these levels i.e. M_0 to M_2 *(e.g. handling class concept when the starting architecture deals with instance concept).*

Abstraction levels (a_0 to a_n). It is often necessary to describe a model at a number of abstraction levels. Human experts use this type of approach to address complex problems. By defining a suitable level of abstraction for a given application phase, we limit the amount of semantic information being considered by relegating irrelevant details to a lower level of abstraction (cf Oussalah, 2014) (*e.g. sub-classes and super-classes are respectively the low and the high abstraction levels for the class concept).*

2.2. Object of evolution (Where?)

The object of evolution is the subject on which the evolution is operated and comprises the following.

Artifact. It is an abstraction of any element belonging to the architectural structure. It can be an architecture, a component, an object, a service or another element.

Process. It is an abstraction for all isolated or aggregate operations and methods employed to carry out evolution; therefore the process also evolves. Hence, it presents suitable means for anchoring the benefits pertaining to cost optimization and quality promotion (cf Pigoski, 1996).

2.3. Type of evolution (What?)

This is the most commonly used dimension when addressing taxonomy issue within software engineering. It is inspired from maintenance typology according to Abran et al. (2001) and Swanson (1976). Another view could also refer to the type

as forms of evolution: curative and anticipative. Figure 1 shows the two main sub-dimensions of the evolution type: main-categories and main-forms.

2.3.1. Main-categories

The main-categories reflect the response regarding a motivation that invokes the evolution which can derive from intentions-based typology (cf Chapin et al., 2001). Herein, we simply employ those commonly used: corrective, perfective and adaptive.

2.3.2. Main-forms

Often, architects predefine the strategy used for the evolution problem-solving according to two forms.

(a) Curative- can be applied in run or load times either when new requirements arise unpredictably or are poorly defined or even unspecified during the life cycle. Meanwhile the environment helps to correct and enhance them by integrating the desired changes, whether permanently or temporarily.

(b) Anticipative- can be applied when evolution requirements are taken into account during the analysis phase and are specified during the design-time or the load-time as well.

According to the literature, these forms can occur in predefinition (cf Chaki et al., 2009) or continuous (cf Oussalah, 1999) fashion. The first of these emphasizes two views.

(a) Open evolution means a deduction, from an initial architecture, of a new architecture reflecting a system solution in which a set of invariants and constraints are respected.

(b) Closed evolution, on the other hand, means the evolution activity tries by induction to find the best permissible sequence of operations to be applied to achieve the desired result. The second, meanwhile, provides:

(c) Evolution with break, which means that interventions are applied directly to the initial architecture without having the ability to go back on the trace of the evolution,

(d) Evolution with seamless denotes an evolution with trace whereby the architecture keeps a trace of its initial properties and operations performed before each evolving process.

For instance, the type of evolution expresses the way of conducting the evolution to implement the requested changes. De facto, it should prescribe both the architectural and technical types of change for embedding predictable or unpredictable requirements.

Main forms			Main-categories		
			Corrective	Perfective	Adaptive
Curative	Open	Break			
		Seamless			
	Close	Break			
		Seamless			
Anticipative	Open	Break			
		Seamless			
	Close	Break			
		Seamless			

Figure 1. Type of evolution structure.

2.4. Operating mechanism of evolution (How?)

By operating mechanism of evolution (OME in the following) we refer to the general behavior employed by only considering the hierarchical levels. Thus, It will inform about time-constraints, properties of changes, impact and propagation of changes invoked in Buckley et *al.* (2005) and Williams et *al.* (2010). Mainly, there are two methods of OME.

Reduce. Brooks (1989) defines the ”reductionist” approach as ”classical” approach to problem solving to which the overall resolution task is decomposed into subtasks. We consider that during the reductionist evolution, the process gets through a predefined evolution path; until the solution is satisfied (evolved model) *(e.g. the evolution of classes impacts the instances).*

Emergence. On the contrary during an ”emergentist” evolution approach, the process builds the path to a solution. Indeed, the emergence expose a passage between the activity of ”micro-level” and that of ”macro-level” *(e.g. categorization of classes in super-classes).*

3. Evolution Taxonomy

Taxonomy provides a common vocabulary for comparison and distinction of the different research studies. A taxonomic class presents a common method of classifying based on the change impact across the architecture modeling and abstraction levels (reduce or emergence). Target architecture can emerge through one of three paths: first, through modification of properties of certain architectural

elements and/or functionalities (referred as 'intra-abstraction level'), second, through derivation of several concepts within the same level of modeling (referred as 'inter-abstraction level'), and last, through various concepts being handled at different modeling levels (referred as 'modeling level'). These three paths are applicable to each metric of OME dimensions, hence the need for six classes for the proposed taxonomy in accordance with the above assumptions (Figure 2).

3.1. Intra-abstraction level reduce-oriented evolution (Class 1)

The evolution process consists of evolving one or more elements of an initial architecture without this having an impact on its modeling level or its abstraction level (*e.g. modification of one or more component properties of an architecture*).

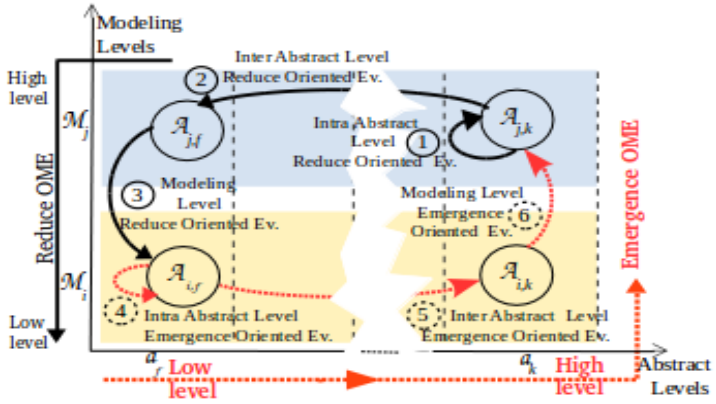

Figure 2. Architecture evolution taxonomy.

3.2. Inter-abstraction level reduce-oriented evolution (Class 2)

The impact of evolution on the architectural elements is improved over several abstraction levels while maintaining the same modeling level of the initial architecture (*e.g. evolution of class impacts several sub-classes*).

3.3. Modeling level reduce-oriented evolution (Class 3)

The architectural elements are enhanced on a downward modeling path through several architectural modeling levels (*e.g. evolution of class impacts instances*).

3.4. Intra-abstraction level emergence-oriented evolution (Class 4)

The evolved architecture is enriched with new architectural elements by applying operations which do not have an impact on levels of abstraction and modeling other than those of the initial architecture (*e.g. categorization of classes by creating a super-class*).

3.5. Inter-abstraction level emergence-oriented evolution (Class 5)

Using the same modeling level as the initial architecture, the emergence of new architectural elements is manifested through the involvement of several levels of abstractions for reasons of simplicity (*e.g. aggregation of classes in one class*).

3.6. Modeling level emergence-oriented evolution (Class 6)

The evolution process implicates several modeling levels on an upward modeling path in order to provide the final architecture with new architectural elements (*e.g. creation of new classes to deal with differences on multiple instances*).

4. Conceptual framework meta-model

4.1. Presentation

A framework meta-model provides an abstract on how its prominent dimensions are organized and composed and thereby it would serve as a meaningful tool for understanding the framework conception. Figure 3 shows the conceptual framework meta-model built around the four dimensions evoked above. These dimensions emerge in response to four major concerns: what, where, who and how software architecture evolves. Besides describing the main forms of resolution strategy: open, closed, seamless or break, the 'What?' concern produces answers regarding the main categories needed for reacting efficiently to the motivation that evokes the change. Meanwhile, 'Where?' informs on objects impacted while the architecture is evolving. Artifact and process are the main objects where the evolution occurs. The third question ('How?') indicates the strategy and the manner in which changes are performed by architects and developers in order to achieve the evolution process. Reduce and emergence are the key insights for solving evolution issues and developing an evolution taxonomy that comprises six classes. Meanwhile, the 'Who?' question identifies the stakeholder hierarchical level affected by the evolution process. Modeling and abstraction levels are the prominent drivers for stakeholder change interventions. It is therefore noteworthy that the relationship between the modeling and abstract sub-dimensions emphasizes the non isomorphism between the two concepts. It indicates that a modeling level can be composed of

several abstraction levels. Here, we underscore that the 'When?' concern was not presented in our conceptual framework for describing the time of evolution (static, dynamic, load-time as in Buckley et *al.* (2005). Nevertheless, the dimension of hierarchical levels can help answer this through deciding whether the evolution occurs at the design-time (from M_1 to M_2 modeling levels) or at the run-time (M_0 modeling level). This concern may lead to further refinements of the proposed conceptual framework that could be taken into account in our future work.

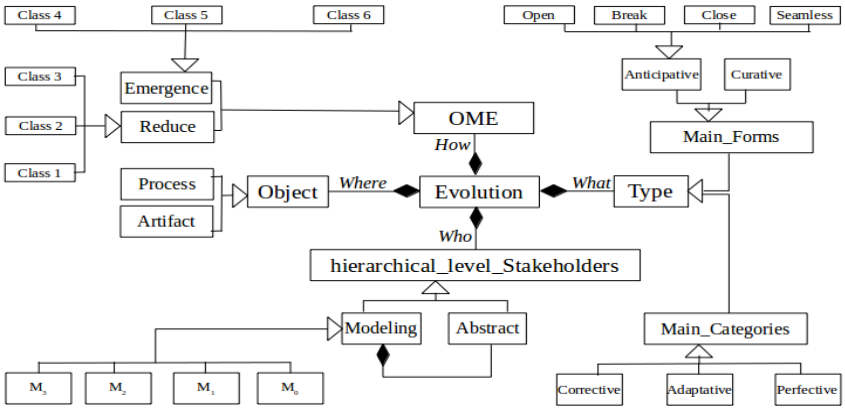

Figure 3. Meta-model of the proposed conceptual framework.

4.2. Application example

In this section, two approaches of architectural evolution (Barnes et al., 2014; Oussalah et al., 2006) are analyzed and compared to show both applicability of our proposal and classification method. The selected approaches are only cited as an example for evolution approaches among many other relevant approaches. Authors in Barnes et al. (2014) provide an automatic approach to assist architects by planning alternative evolution paths for evolving software architecture. An evolution path is expressed in terms of intermediate architecture generated from the initial state. If the architect's goal is clear, this approach aims to assist architects to find the optimal path that meets the evolution requirements. The evolution path acts explicitly on software architecture and applies evolution from a higher to lower level to reach the desired architecture. This approach helps architects to correct and perform the adequate path for the solution. In Oussalah et al. (2006), Software architecture evolution model (SAEV) aims to describe and manage the evolution through the architectural elements of a given software architecture. SAEV considers software architecture elements (component, connector, interface and configuration) as first class entities. Managing these elements is conceived independently of any architecture language level by considering the different levels of modeling (meta level, architectural level and application level). Applying the conceptual framework to the two approaches, Table 1 quantifies all dimensions using three values 1, 0.5

and 0 which reflect respectively explicit, implicit and not mentioned. Thus, this quantification of a given approach enables it to be assigned to a specific class of our taxonomy.

Table 1. Comparison of two evolution approaches using the framework.

Model	Level		Object		OME		Type						
							Form				Category		
	M	A	A	P	R	E	OB	OS	CB	CS	C	P	A
Oussalah (2006)	1	0.5	1	0	1	0	1	0	0	0	1	0	0
	75%		50%		50%		12.50% = (1/4)/2				16.67%= (1/3)/2		
Barnes (2014)	0	0.5	1	0	0.5	0	0	0	1	0	0.5	0.5	0
	25%		50%		25%		12.50%				16.67 %		

Level: (M)odeling, (A)bstraction; Object: (A)rtifact, (P)rocess; OME: (R)educe;(E)mergence, Form:(O)pen, (C)lose, (B)reak, (S)eamless; Category:(C)orrective, (P)erfective, (A)daptive.

5. Conclusion

This research study exposes a taxonomy for classifying evolution architectural approaches through a conceptual framework based on four dimensions which are considered as the main axes that foster various solutions for architecture evolution models. These dimensions are usually expressed explicitly or implicitly as properties or concepts but not as entities to quantify. An application example is carried out to demonstrate how such dimensions can be quantified, and then provide help on the approach classification. Considering impacts of evolution process dimension on hierarchical (modeling/abstraction) levels we have formalized an evolution taxonomy which sketches the interior design of evolution solution space. Furthermore, in the present study, we record certain limitations due to the subjective estimation using three quantitative values (explicit, implicit and not mentioned). Thus, we believe that the conceptual framework that we have presented, could:(i) compare different architectural evolution approaches, (ii) determine the appropriate evolution approach according to the dimensions that seems the most relevant to the application class and (iii) to guide thinking within research teams on new architectural evolution approaches.

Bibliography

Abran, A., Bourque, P., Dupuis, R., & Moore, J. W. (2001). Guide to the software engineering body of knowledge-SWEBOK. IEEE Press.

Ahmad, A., Jamshidi, P., & Pahl, C.(2014). Classification and comparison of architecture evolution reuse knowledge- a systematic review. Journal of Software: Evolution and Process Vol. 26, n° 7, p. 654-691.

Barnes, J. M., Garlan, D., & Schmerl, B. (2014). Evolution styles: foundations and models for software architecture evolution. Software & Systems Modeling Vol. 13, n° 2, p. 649-678.

Bézivin, J. (2003). La transformation de modèles. INRIA-ATLA S & Université de Nantes, École d'Été d'Informatique CEA EDF INRIA 2003, cours #6.

Breivold, H. P., Crnkovic, I., & Larsson, M. (2012). A systematic review of software architecture evolution research. Information and Software Technology, Vol. 54, n° 1, p. 16-40.

Brooks, R. A. (1989). A robot that walks, emergent behaviors from a carefully evolved network. Neural computation, Vol. 1, n° 2, p. 253-262.

Buckley, J., Mens, T., Zenger, M., Rashid, A., & Kniesel, G. (2005). Towards a taxonomy of software change. Journal of Software Maintenance and Evolution: Research and Practice , Vol. 17, n° 5, p. 309-332.

Chaki, S., Diaz-Pace,A., Garlan, D., Gurfinkel, A., & Ozkaya, I. (2009). Towards engineered architecture evolution. In Proceedings ICSE Workshop on Modeling in Software Engineering, p. 1-6.

Chapin,N. , Hale, J. E., Khan, K. M., Ramil, J. F., & Tan, W. G. (2001). Types of software evolution and software maintenance. Journal of software maintenance and evolution: Research and Practice Vol. 13, n° 1, p. 3-30.

Jazayeri, M. (2005). Species evolve individuals age. In: Principles of Software Evolution. 8th International Workshop on Principles of Software Evolution IEEE, p. 3-9.

Johnson, R. E. (1992). Documenting frameworks using patterns. In ACM Sigplan Notices, Vol. 27, n° 10, p. 63-76.

Oussalah, M.C. 2014. *Software Architecture 1*. Wiley-ISTE, p. 244-246.

Oussalah, M., Sadou, N., & Tamzalit, D. (2006). SAEV:A model to face evolution problem in software architecture. In: Proceedings of the Int. ERCIM Workshop on Software Evolution, pp. 137-146.

Oussalah, M. C. (1999). *Génie objet: analyse et conception de l'évolution*. Hermès science publications.

Pigoski, T. M. (1996). Practical software maintenance: best practices for managing your software investment. John Wiley & Sons Inc.

Shaw, M., DeLine, R., Klein, D. V., Ross, T. L., Young, D. M.,& Zelesnik, G. (1995). Abstractions for software architecture and tools to support them. IEEE Transactions on Software Engineering, Vol. 21, n° 4, p.314-335.

Swanson, E. B. (1976). The dimensions of maintenance. In: Proceedings 2nd International Conference on Software engineering. IEEE Computer Society Press, p. 492-497.

Williams, B. J., & Carver, J. C. (2010). Characterizing software architecture changes: A systematic review. Information and Software Technology, Vol. 52, n° 1, p.31-51.

AgBig : Un modèle basé Bigraphe pour la Spécification des Services Elastiques du Cloud

Khaled KHEBBEB[1], Billel SEGHIR[1], Faiza BELALA[1], Meriem BELGUIDOUM[1]

1. Laboratoire LIRE, Université Constantine 2 - Abdelhamid Mehri, Algérie
{khaled.khebbeb,seghir.billel}@gmail.com,
{faiza.belala, meriem.belguidoum}@univ-constantine2.dz

RESUME. AgBig est un modèle adapté à la modélisation des systèmes évolutifs ubiquitaires, il se base sur les Systèmes Réactifs Bigraphiques pour la représentation des aspects architecturaux (localité et connexions) et leurs évolutions d'une part, et les agents pour la prise en charge de la logique métier et du comportement dynamique et contextuel d'applications évolutives d'une autre. Nous montrons à travers cet article, l'application de ce modèle formel proposé au domaine du Cloud Computing pour la spécification du comportement élastique horizontal des applications au niveau SaaS. Ce comportement est défini par une séquence de reconfigurations architecturales supportées par des règles de réaction étendues. Ces règles sont pilotées par les entités d'agents qui fonctionnent selon le principe Observation-Communication-Action. Par conséquent, l'utilisation de ce type de règles de réaction, contrairement aux règles ordinaires librement instanciées et indépendantes de tout contexte d'utilisation, permet de décrire le comportement dynamique en termes de réécriture de bigraphes prenant en charge les aspects de synchronisation et effets de bord entre les différents états du système.

ABSTRACT. AgBig is a suitable model for formalizing evolutionary and ubiquitous systems. On the one hand, AgBig is based on the Bigraphical Reactive Systems to represent architectural aspects (locality and connectivity) and their evolutions; on the other hand, it uses agents to handle the dynamic and contextual behavior of scalable applications. In this paper, we show how we will apply this model to the Cloud Computing paradigm, in order to specify the elastic behavior of services in the SaaS model. This behavior is defined as a sequence of some architectural reconfigurations supported by a set of extended reaction rules. These rules are piloted by agent entities that operate according to an Observation - Communication - Action principle. This type of reaction rules, contrary to the ordinary ones freely instantiated and independent of any context of use, allows to describe clearly the dynamic behavior of this system in terms of bigraphs rewritings, taking into account synchronization and side effects of system states.

CAL 2016 : 10éme Conférence Francophone sur les Architectures Logicielles.
7-8 Juin 2016, Besançon, France.

MOTS-CLES: Cloud Computing, Service élastiques, Systèmes Réactifs Bigraphiques, Systèmes Multi Agents

KEYWORDS: Cloud Computing, Elastic services, Bigraphical Reactive Systems, Multi Agents Systems

1. Introduction

Le Cloud Computing (Mell et Grance, 2011) a émergé en tant que paradigme informatique prometteur durant ces dernières années. Ce nouveau modèle de livraison est basé sur une idée simple qui consiste à fournir un ensemble de ressources virtualisées (par exemple des serveurs, des machines virtuelles, des conteneurs, des applications, des services, etc.) comme services informatiques à la demande d'une manière élastique. Ces services sont offerts selon trois modèles de services fondamentaux: l'infrastructure en tant que service (IaaS), la plate-forme en tant que service (PaaS), et le logiciel en tant que service (SaaS). En fait, le Cloud Computing est devenu populaire auprès des entreprises qui tendent à utiliser les infrastructures et plateformes Cloud pour déployer, héberger et exécuter leurs services. Cette popularité, de plus en plus croissante, est due à ses nombreux avantages tels que la mobilité, la flexibilité, la rentabilité et de façon plus significative, la propriété d'élasticité. L'élasticité représente l'une des principales qualités attendues d'un système Cloud, son but est de préserver la qualité de service (QoS) par la mise à l'échelle en fonction de la charge de travail, aussi bien à la hausse qu'à la baisse de celle-ci; évitant ainsi le sous/sur provisionnement des ressources. De plus, l'élasticité est très importante pour la sécurité du Cloud. Elle peut jouer un rôle important dans la lutte contre les attaques basées sur le Web telles que le déni de service distribué, *Distributed Denial of Service* (DDoS), où les attaquants peuvent essayer de consommer toutes les ressources disponibles en accablant les mécanismes d'élasticité (Bauer et Adams, 2012). Selon la classification de (Galante et Bona, 2012), l'élasticité peut être assurée au moyen de trois méthodes fondamentales: la mise à l'échelle *horizontale* ou *verticale* et la *migration* (Coutinho *et al.*, 2014). La méthode de mise à l'échelle horizontale consiste à ajouter ou à retirer les instances d'un service en fonction des variations de la charge de travail. L'approche de mise à l'échelle verticale (redimensionnement) consiste à rajouter ou à supprimer des ressources virtualisées (par exemple la mémoire, CPU, stockage, etc.) à partir d'une machine virtuelle lors de l'exécution. Enfin, la méthode de migration est la relocalisation d'une machine virtuelle en cours d'exécution (ou d'un service dans certains cas) à partir d'un serveur physique à un autre. Dans ce travail nous nous concentrons principalement sur la fourniture de l'élasticité horizontale au niveau du modèle de service SaaS.

Nous visons, à travers cet article, à proposer un modèle formel pour la spécification du comportement élastique des services Cloud. Les méthodes formelles fournissent une base mathématique fiable assurant des modèles analysables et facilement vérifiables. Ainsi, nous définissons un modèle formel combinant les Systèmes Réactifs Bigraphiques ou BRS pour *Bigraphical Reactive Systems* de (Milner, 2008) et le concept des Systèmes Multi-Agents (Sycara, 1998**)** pour

spécifier l'élasticité des services Cloud. Cette combinaison peut constituer une solution intéressante pour représenter d'une manière convenable l'élasticité des services Cloud. En effet, les BRS sont adoptés pour leur aspect graphique, leur base formelle rigoureuse et leur capacité de représentation de la localité et l'interaction des services entre eux, tandis que les agents fournissent un moyen intelligent pour assurer l'élasticité au niveau des services.

La suite de l'article est organisée comme suit. La section 2, présente quelques recherches jugées pertinentes pour notre travail. La section 3 présente les concepts de base des systèmes réactifs bigraphiques (BRS) et les systèmes multi agents (SMA). Le modèle proposé est présenté dans la section 4 puis illustré à travers un exemple dans la section 5. Enfin, la section 6 conclut le papier et discute des travaux futurs.

2. Travaux connexes

Durant les dernières années, la spécification et l'analyse formelles de l'élasticité dans les environnements Cloud Computing sont devenues un sujet de recherche d'une importance de plus en plus grandissante comme dans les travaux de (Sahli *et al.*, 2014a ; 2014b ; 2015), de (Gambi *et al.*, 2013) et de (Bersani *et al.*, 2014). Cependant, il existe peu de travaux de recherche dans la littérature qui traitent la formalisation de l'élasticité dans le Cloud au niveau service (SaaS). Dans ce contexte, les auteurs (Amziani *et al.*, 2013a) introduisent un cadre formel basé sur les réseaux de Petri pour la description et l'évaluation de l'élasticité de processus métier exprimés en tant que services; ceci a été accompli par la définition de deux réseaux de Petri pour les opérations de duplication et de consolidation relatives à la mise à l'échelle horizontale. Une extension de cette approche pour soutenir l'évaluation basée sur le temps a été proposée dans les travaux de (Amziani *et al.*, 2013b**)** en utilisant les réseaux de Petri colorés (CPN). D'un autre côté, les systèmes réactifs bigraphiques de Milner ont déjà été révélés très utiles dans la formalisation de systèmes distribués, sensibles au contexte et ubiquitaires comme dans les travaux de (Sevegnani et Pereira, 2011) et de (Wang *et al.*, 2011) ainsi que d'autres utilisations dans d'autres domaines différents, notamment pour la modélisation de systèmes multi-agents par des BRS (Mansutti *et al.*, 2014), nos travaux se distinguent de ceux là dans la mesure où nous proposons la combinaison des BRS et des SMA pour élaborer une approche de modélisation adaptée à des systèmes distribués ubiquitaires.

Nous soutenons que la localité et la connectivité sont des aspects clés du Cloud Computing; ce qui nous permet d'avancer que les BRS sont une solution idéale pour modéliser les services Cloud et de leurs différents types d'interactions par le biais des règles de réaction des BRS, alors que les SMA présentent une façon intelligente pour assurer l'élasticité au niveau des services. L'idée de combiner des systèmes réactifs bigraphiques avec les systèmes multi-agents figurent dans quelques travaux de recherche. Par exemple, les travaux de (Pereira *et al.*, 2012) intègrent les BRS avec des agents génériques afin d'introduire BiAgents, un modèle de calcul concurrent pour la modélisation de systèmes informatiques mobiles. Dans ces

travaux, l'approche agent n'est pas entièrement exploitée, les auteurs ont plus recours à une philosophie agent qu'au paradigme agent. Par exemple, les notions de communication et d'échanges entre agents en sont absentes. Notre modèle tentera de palier à ce manque.

3. Notions de base

Dans cette section, nous présentons brièvement les concepts de base des systèmes réactifs bigraphiques et des systèmes multi-agents.

3.1. Les systèmes réactifs bigraphiques

Les Bigraphes de Milner (Milner, 2008) réunissent deux structures indépendantes : le graphe de places (*place graph*) et le graphe de liens (*link graph*), d'où le nom Bigraphe. Le graphe de places représente la distribution géographique des entités du système. Le graphe de liens représente des interactions entres ces entités. Les Bigraphes sont accompagnés par des règles de réaction définissant leurs dynamiques structurelles. La figure 1 illustre l'anatomie d'un bigraphe.

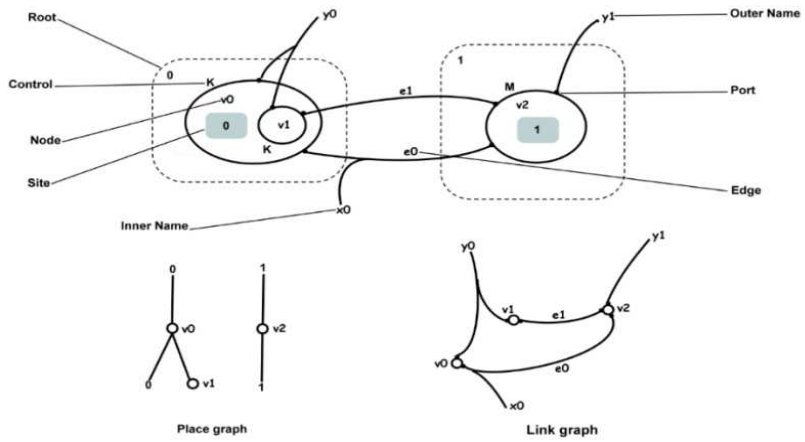

Figure 1. Anatomie d'un bigraphe

Une région (Root) sert à identifier la partie d'un Bigraphe pouvant être hébergée, elle peut contenir un ensemble de nœuds. Un nœud (Node) est un composant physique du système. Les nœuds peuvent être imbriqués ou situés à l'intersection d'un ou de plusieurs nœuds. Chaque nœud peut avoir un ou plusieurs ports pour représenter les éventuelles connexions. Un port, spécifique à un nœud, est un point de connexion avec les autres entités du Bigraphe. Les liens (Edge) servent à relier

deux ou plusieurs ports. Un site est une partie du modèle susceptible d'héberger une région du Bigraphe. Les interfaces entrantes/sortantes (Inner/Outer Name) sont utilisées pour indiquer des liens potentiels à d'autres bigraphes. Un Bigraphe supporte une certaine dynamique, non pas comportementale mais structurelle, il passe alors d'un cliché à un autre (d'une configuration à une autre) à la rencontre de conditions connues, appelées "Règles de réaction". Une règle de réaction est une application notée $\boldsymbol{r \rightarrow r'}$ (Milner, 2009) qui sert à transformer un Bigraphe *Redex* en un Bigraphe *Reactum*. La figure 2 illustre un exemple d'application d'une règle de réaction mettant en scène un nœud *P* hébergé par un nœud *R* qui lui-même est hébergé par une région 0. La règle de réaction consiste en la modification de la localisation du nœud *P* de manière à définir son hébergement par la région 0 et non plus par le nœud *R*. Cette règle modifie donc le graphe de place lié à ce bigraphe et n'apporte aucune modification quant au graphe de lien.

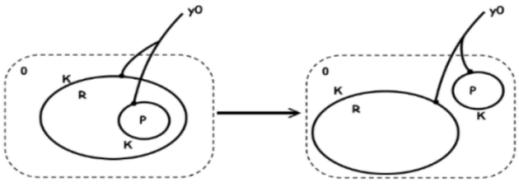

Figure 2. Exemple de règle de réaction

3.2. Les systèmes multi-agents

Un système multi-agents est considéré comme un ensemble d'agents intelligents interagissant entre eux sur un environnement commun et ayant la capacité de coopérer, coordonner, et négocier avec un autre en vue de résoudre un problème ou de réaliser un but spécifique à travers leurs capacités de raisonnement individuelles (Sycara, 1998). Un agent est une entité autonome située dans un environnement virtuel ou physique et capable d'y opérer. Les agents peuvent être classés en trois catégories principales : réactifs, cognitifs et hybrides.

Dans ce travail, nous utilisons un ensemble d'agents réactifs qui se comportent suivant une approche stimuli-réponse dans le but de refléter le comportement élastique au niveau service dans un environnement Cloud Computing exprimé en termes de bigraphes.

4. Approche de modélisation

Le modèle AgBig peut servir pour la modélisation des aspects architecturaux (déploiement) et comportementaux (dynamique) des systèmes évolutifs. La couche physique est exprimée par les bigraphes et la couche logique par les agents, de telles entités apportent des mécanismes d'*observation*, de *communication* et d'*action*

offrant un comportement aux entités physiques du système caractérisé par le guidage de leur restructuration dans un sens contextuel. Les entités agents déployées au sein du bigraphe, à la rencontre de situations données qu'elles identifient en observant leur environnement, collaborent et communiquent entre elles dans le but d'adapter le système (le bigraphe) à des situations contextuelles. Ce, en déclenchant une ou plusieurs règles de réaction guidant ainsi la restructuration du bigraphe. Nous considérons que ces mécanismes mettent en œuvre ce que nous supposons être la dynamique comportementale des systèmes modélisés par AgBig.

4.1. Définition formelle

Le modèle AgBig est défini par la structure *(B, A, Host, M, Act)* donnée par:

– $B= (V_B, E_B, ctrl, G^P, G^L) :<m, X> \rightarrow <n, Y>$ est le bigraphe représentant l'aspect physique du système modélisé (Milner, 2009), où :

- V_B désigne l'ensemble des nœuds du bigraphe *B* représentant les différentes entités considérées dans un système donné;

- E_B est l'ensemble des liens définissant toutes les liaisons port à port entre ces entités;

- $ctrl : V_B \rightarrow K$ est une fonction qui associe à chaque nœud, son contrôle, i.e. son type et le nombre de ses ports;

- G^P est le graphe des places, il indique la localisation des entités du système dans l'espace;

- G^L est le graphe des liens, il identifie les différentes connexions qui existent entre les entités du système à modéliser (interfaces d'entrée/sortie et autres);

- *X* et *Y* définissent les interfaces d'entrée et de sortie;

- *m* et *n* modélisent respectivement le nombre de mémoires libres (sites) et le nombre de serveurs (régions).

– *A* est l'ensemble d'agents opérant sur *B*;

– $Host : A \rightarrow V_B$ est la relation d'hébergement d'un agent par un nœud du bigraphe;

– *M* est l'ensemble des messages m_i échangés (émis et reçus) par *A*;

– *Act* est l'ensemble de types d'actions *Action.* Une action $Action : A^* \rightarrow M$ représente une opération générique appliquée aux instances d'agents en argument et qui retourne un message.

4.2. Modélisation des systèmes Cloud dans AgBig

La modélisation à l'aide de AgBig peut constituer une solution aux problèmes internes courants liés à l'exécution des systèmes évolutifs; elle permet de proposer une représentation des comportements basés sur les agents pour gérer d'une manière efficace l'élasticité horizontale des services dans le Cloud, ce, en pilotant l'application des règles de réaction du bigraphe modélisant le système Cloud d'une

manière contextuelle. Le tableau 1 montre les correspondances entre les différents concepts d'un système Cloud et leur représentation dans le modèle AgBig.

Tableau 1. Correspondances entre concepts du Cloud et éléments AgBig

Concept du Cloud	Élément du modèle AgBig
Serveur	Région r_i
Service	Nœud de V_B
Interactions entre services	Lien de E_B
Demande de service	Nom interne X
Envoi de service	Nom externe Y
Ressource disponible	Site s_i
Relation entre le service et son état	Host(A_i)
État d'un service observable	instance d'un agent de A
Effet de bord (changement contextuel d'un service)	Message m_i de M
Déclencheur d'un changement d'état	Action de *Act* telle que *envoie, réagit.*
Trace d'exécution	Règle de réaction étendue de type $R_i : B_i \rightarrow^{Action(Am,An,\ldots)} B_j$

Le modèle introduit propose une séparation entre les aspects physiques et logiques d'un système. Suivant le premier aspect, un système Cloud est vu comme un bigraphe réunissant un nombre donné de services (nœuds) distincts. Ces services sont déployés dans un serveur (région) ou distribués sur multiples serveurs (plusieurs régions), ils interagissent les uns avec les autres à travers des liens. Un système Cloud peut offrir ou requérir des services du monde extérieur (autres systèmes) respectivement via les noms internes et externes du bigraphe le modélisant. Les ressources disponibles prévues pour l'accueil d'instances de services sont exprimées par des sites du bigraphe. Le second aspect de modélisation concerne la partie logicielle (état) du système exprimée par les entités d'agents, la relation entre un service et son état (agent) est vue comme une relation d'hébergement de l'agent adéquat par le nœud concerné. L'exécution d'un service peut affecter tout le système. Nous désignons par le terme ''effet de bord'', les événements qui occurrent au sein d'un service et qui suscitent une prise en charge par l'environnement ce qui peut être traduit de la sorte : les agents déployés dans le système échangent des messages et entreprennent des actions (comme des envois de messages ou des réactions) dans le but de gérer un évènement ou un changement donné. Une réaction peut être vue comme une trace d'exécution, i.e., une ou plusieurs règles de réaction implémentant l'action visée par la coopération entre les agents.

5. Étude de cas

Pour présenter notre approche, nous proposons une étude de cas inspirée de l'exemple de (Klai et Tata, 2013) présentant un service de vente d'ordinateurs en ligne composé de quatre services: le service de requêtes (*S1*) reçoit les requêtes d'achat d'un ordinateur (temps de traitement = 1s); le service d'assemblement de composants (*S2*) effectue l'assemblage des composants d'ordinateurs en fonction des caractéristiques désirées par les clients (temps de traitement = 60s); le service de facturation (*S3*) crée la facture liée à l'ordinateur acheté (temps de traitement 1s); le service de livraison de la facture (*S4*) délivre la facture au client (temps de traitement = 1s). Le temps de traitement d'une seule requête sera de 62s.

Le diagramme d'activité en figure 3 illustre l'ordre d'exécution des sous-services, le service *S1* s'occupe de déléguer les requêtes aux services *S2* et *S3* qui opèrent de manière simultanée (parallèle). Le service *S4*, une fois la requête traitée par *S2* et *S3*, la traite pour la phase finale du processus.

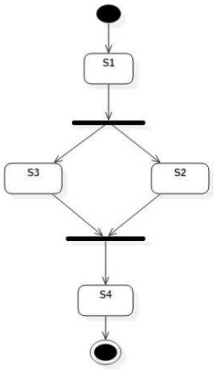

Figure 3. Diagramme d'activité du service d'achat en ligne

Supposons que ce système soit sujet à une attaque Web de catégorie DDoS *(Distributed Denial of Service)* (Bauer et Adams, 2012) et qu'il reçoive 100 requêtes simultanément. Ce type d'attaque Web consiste à solliciter le système à partir de centaines de sources différentes à la fois et peut rapidement surcharger le système, voire le rendre complètement indisponible si aucune méthode d'élasticité n'est adoptée. Dans ce cas le temps de réponse du système serait égal à 6200s (62s x 100 requêtes).

Une solution pour améliorer ce temps de réponse serait de dupliquer le système pour supporter la charge. Donc, pour traiter 100 requêtes, nous pourrions avoir 100 copies du système déployées en parallèle (une requête pour chaque instance du système). Le temps de traitement des 100 requêtes serait alors égal à 62s. Cette

solution résout le problème de temps de réponse mais en pose un autre concernant la consommation de ressources. En effet, le désavantage de cette solution est la duplication de tous les services, ce qui n'est pas toujours possible et dépend de la disponibilité des ressources nécessaires au niveau de l'environnement de déploiement du système. Nous estimons qu'il n'est pas nécessaire de dupliquer tout le système et donc tous les sous-services le constituant du moment que le "goulot d'étranglement" est provoqué par un seul service (S2). En effet, la duplication du service ralentissant le fonctionnement du système résoudrait le problème de temps de réponse tout en évitant la consommation inutile de ressources.

Nous proposons de recourir à la méthode de mise à l'échelle horizontale pour la gestion de l'élasticité de ce système. Dans notre exemple, en comparaison avec les services *S1, S3* et *S4*, le service *S2* est le plus gourmand en ressources consommées. Nous considérons qu'en termes d'élasticité, dupliquer (ajouter) ou consolider (supprimer) des instances de *S2* est suffisant.

Imaginons maintenant que ce système soit proposé comme un service Cloud au niveau SaaS. Dans la représentation de ce système par le modèle AgBig, nous pouvons, par exemple, choisir de disposer les services de la manière suivante: les services *S1, S3* et *S4* seront déployés au sein d'un même serveur car ne font pas l'objet de mécanismes d'élasticité. Le service *S2*, doté de comportement élastique, sera déployé au sein d'un serveur séparé. Ce choix peut être fait dans le but d'optimiser la consommation de ressources entre les deux serveurs. La figure 4 illustre la représentation de ce système dans le modèle AgBig. La région *r0* représente le serveur hébergeant les services *S1, S3 et S4* tandis que la région *r1* représente le serveur hébergeant le service élastique *S2*. Le site c1 présent sur la région *r1* représente la disponibilité de ressources pouvant accueillir une nouvelle instance du service *S2*.

Le comportement élastique (duplication) de *S2* est piloté par les agents *Agent1* (hébergé par *S1*) et *Agent2* (hébergé par *S2*) selon le scénario suivant: (1) *S1* reçoit une nouvelle requête, celle-ci est redirigée simultanément vers *S2* et *S3*. (2) Une autre requête parvient à *S1* et l'instance de *S2* déjà déployée est occupée (3) si un site est disponible au niveau de la région *r1* (4) l'agent *Agent1* prend connaissance du point 3 par observation et (5) provoque par action l'application d'une règle de réaction étendue qui consiste en la création d'un nouveau nœud *S2'* au niveau du site libre puis sa liaison avec *S1* et *S4*. La figure 5 illustre l'état du système après ces étapes d'exécution qui sont schématisées par les réécritures suivantes. Sachant que chaque réécriture peut appliquer 0 ou plusieurs règles de réaction ordinaires guidées par les communications inter agents.

$$B0 \rightarrow^{observe(Agent1)} B0 \rightarrow^{réagit(Agent1)} B1$$

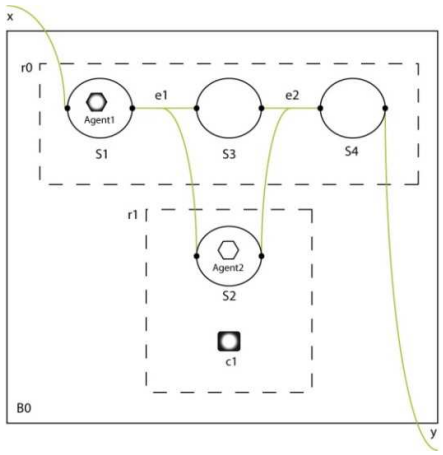

Figure 4. Représentation du service d'achat en ligne dans AgBig

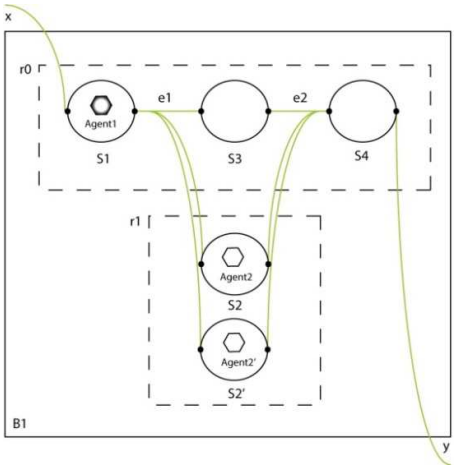

Figure 5. Duplication du service S2

Remarque: le processus de consolidation est guidé par l'agent Agent2 qui, une fois le traitement de son nœud hôte (S2 par exemple) terminé, envoie par communication un message de notification de la situation à Agent1 qui provoque une règle de réaction étendue consistant à supprimer S2 et ses connexions. La réécriture associée à ce mécanisme est la suivante, B2 étant le bigraphe résultant de la réécriture. $B1 \rightarrow^{envoie(Agent2,Agent1)} B1 \rightarrow^{réagit(Agent1)} B2$

6. Conclusion

Le modèle AgBig est une combinaison judicieuse de BRS et Systèmes Multi-Agents adoptée comme cadre sémantique pour la spécification des systèmes évolutifs. D'un coté, les bigraphes servent à exprimer la localisation et les interconnexions des entités constituant les systèmes modélisés. De l'autre, les règles de réaction et les entités agent fournissent au bigraphe la capacité d'auto-reconfiguration d'une manière tant structurelle que comportementale. En effet, le modèle AgBig associe des classes d'agents au comportement des bigraphes. Ce comportement est exprimé comme une suite observation-communication-action menée par les agents pour provoquer des règles de réaction étendues définissant la dynamique comportementale du bigraphe.

Le modèle s'est avéré très utile dans la formalisation de la dynamique des systèmes Cloud, particulièrement pour traiter l'élasticité horizontale des services Cloud au niveau SaaS. Comme travaux futurs, nous comptons raffiner le modèle AgBig proposé pour prendre en considération les autres techniques assurant le caractère élastique d'un service Cloud (mise à l'échelle verticale et migration) ainsi que son processus de déploiement.

D'autre part, nous prévoyons de développer un outil d'édition de modèles AgBig permettant la génération automatique des spécifications correspondantes dans Maude (Clavel *et al., 2012),* une plateforme pour la spécification et vérification formelles et exécutables des systèmes concurrents et distribués, afin d'exploiter notre approche proposée et de vérifier d'autres propriétés inhérentes du Cloud Computing.

Bibliographie

Amziani, M., Klai, K., Melliti, T., Tata, S. (2013), Time-based evaluation of service based business process elasticity in the Cloud. In: IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 573 580.

Amziani, M., Melliti, T., Tata, S. (2013), Formal modeling and evaluation of service based business process elasticity in the Cloud. In: 22nd IEEE International Conference on Collaboration Technologies and Infrastructure (WETICE 2013), pp. 284-291. Hammamet, Tunisia.

Bauer, E., Adams, R. (2012), Reliability and availability of Cloud Computing. Wiley IEEE Press, first edition.

Bersani, MM., Bianculli, D., Dustdar, S., Gambi, A., Ghezzi, C., Krstić, S. (2014) Towards the formalization of properties of Cloud-based elastic systems. In: Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems (PESOS 2014), pp. 38-47. ACM, New York, USA.

Clavel, M., Duran F., Eker S., Lincoln P., Marti-Oliet N., Meseguer J., and Quesada J. (2002), Maude: Specification and programming in rewriting logic. Theoretical Computer Science, 285 :187–243.

Coutinho, E., de Carvalho, S.F., Rego, P., Gomes, D., de Souza, J. (2014) Elasticity in Cloud Computing: a survey. In: annals of telecommunications issn: 0003-4347, pp. 1-21. Springer, Heidelberg.

Galante, G., de Bona, L. (2012) A survey on Cloud Computing elasticity. In: proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing (UCC'12), pp. 263-270. IEEE Computer Society, Washington, USA.

Gambi, A., Filieri, A., Dustdar, S. (2013) Iterative test suites refinement for elastic computing systems. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013), pp. 635-638. ACM, New York, USA.

Klai, K. and Tata, S. (2013). Formal Modeling of Elastic Service-based Business Processes. In: Proc. 2013 IEEE 10th International Conference on Services Computing.

Mansutti, A., Miculan, M., Peressotti, M. (2014), Multi-agent systems design and prototyping with bigraphical reactive systems. In: Distributed Applications and Interoperable Systems, pp. 201-208. Springer, Heidelberg.

Mell, P., Grance, T. (2011), The NIST Definition of Cloud Computing, National Institute of Standards and Technology Special Publication 800-145 7 pages.

Milner, R. (2008), Bigraphs and their algebra. In: Proceedings of the LIX Colloquium on Emerging Trends in Concurrency Theory, Electronic Notes in Theoretical Computer Science, Volume 209, pp. 5-19. Elsevier.

Milner, R. (2009), The space and motion of communicating agents. Cambridge University Press (200 pages).

Pereira, E., Kirsch C. and Sengupta R (2012). Biagents - A Bigraphical agent model for structure-aware computation. Working Papers CPCC-WP-2012-08-01.

Sahli, H., Belala, F., Bouanaka, C. (2014a), Model-checking Cloud systems using BigMC. In: proceedings of the 8th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS 2014), pp. 25 33. Bejaïa, Algeria.

Sahli, H., Belala, F., Bouanaka, C. (2015), A BRS-Based Approach to Model and Verify Cloud Systems Elasticity. HOLACONF - Cloud Forward: From Distributed to Complete Computing. Procedia Computer Science 68 (2015) 29 – 41.

Sahli, H., Bouanaka, C., Dib, AT. (2014b), Towards a formal model for Cloud Computing elasticity. In: IEEE 23rd International WETICE Conference (WETICE), pp. 359 364. Parma, Italy.

Sevegnani, M., Pereira, E. (2011), Towards a bigraphical encoding of actors. In: T. T. Hildebrandt, editor, Proc, MeMo (2014) 13. Wang, J., Xu, D., Lei, Z.: Formalizing the structure and behaviour of context-aware systems in bigraphs. In: First ACIS International Symposium on Software and Network Engineering.

Sycara, K.P. (1998), The many faces of agents. AI Magazine, 19(2), pp. 11-12.

Wang, J., Xu, D., Lei, Z. (2011), Formalizing the structure and behaviour of context-aware systems in bigraphs. In: First ACIS International Symposium on Software and Network Engineering.

Vers une définition formelle du Cloud Computing Mobile

C. Mahmoudi[1], F. Mourlin[1], G. L. Djiken[1], Y. Atik[2]

1. Laboratoire d'algorithme, Complexité et Logique, Université de Paris Est-Créteil
61 avenue du Général de Gaulle, 94010 Créteil Cedex, France
charif.mahmoudi@lacl.fr , fabrice.mourlin@u-pec.fr, guy-lahlou.djiken@u-pec.fr,

2. Quattro-IT
8 Place porte de Champerret, 75017 Paris
quattro.it2@gmail.com

RÉSUMÉ. Dans le domaine émergent du Mobile Cloud Computing (MCC), 2 tendances se réunissent pour composer les principaux piliers. D'un côté nous avons les tendances de virtualisation qui affectent les Datacenters et d'autre part, nous avons les appareils mobiles qui se sont révélés être des outils les plus efficaces et pratiques. Au niveau du Cloud Computing, le développement porte autour de la mobilité surtout en termes d'espaces de travail, d'interaction avec les périphériques connectés et des capteurs. Ce document fournit une spécification formelle écrite avec le langage π-calcul qui définit la représentation des périphériques virtuels dans le MCC. En outre, nous décrivons une nouvelle vision des périphériques virtuels composites utilisés par tous les périphériques, les capteurs et les actuateurs disponibles sur le réseau. Nous abordons la déportation de l'application et la gestion en réseau des périphériques virtuels sur des Clouds mobiles. Notre modèle architectural est inspiré du système de base de Cloudlet. Nous présentons nos spécifications proposées, l'architecture du modèle, ainsi qu'un cas d'étude montrant la congruence structurelle entre une application exécutée localement et une version déportée de cette même application.

ABSTRACT. In the emerging area of mobile cloud computing, 2 tendencies come together to compose some of the major pillars. On one hand we have the virtualization trends that are affecting the data centers and on the other hand we have mobile devices which proved to be the most effective and convenient tools. This emerging area is then changing the mobility of workspaces and the interaction with the connected devices and sensors. This paper provides a formal specification using the π-calculus language. It defines the virtual device representation in the mobile cloud computing. In addition, we describe a new vision of composite virtual devices that takes advantage of all devices, sensors, and actuators available on the network to build a composite virtual device. We address application offloading and virtual devices networking on mobile clouds. Our architectural model comes from a Cloudlet based system. We introduce our proposed specifications, architecture, along with a case study showing the structural congruence between a locally executed application and an offloaded application.

MOTS-CLES : Définition formelle, migration, mobile, Cloud Computing Mobile, Offloading, Virtualisation, Représentation de périphérique mobile.

KEYWORDS: Formal definition; migration; mobile; mobile cloud computing; offloading; virtualization; virtual device representation.

1. Introduction

De nos jours, les périphériques mobiles sont de plus en plus utilisés, essentiellement comme outil de communication le plus efficace et pratique. La non limitation du temps et d'espace d'utilisation de ces périphériques permet aux utilisateurs mobiles d'accumuler une riche expérience de ses divers services et applications. L'exécution de ces services ne se limite pas au périphérique mobile lui-même. Les architectures basées sur le N-tiers computing se sont répandues exponentiellement dans le développement des technologies de l'information (IT ou Information Technology) ainsi que dans les domaines du commerce et de l'industrie pour l'informatique mobile [1]. Cependant, les appareils mobiles ont des limites fortes au niveau du matériel. L'informatique mobile fait face à des défis en essayant de fournir les diverses applications résidant dans un seul périphérique avec des ressources limitées tels que la batterie, la mémoire et la bande passante. Ces défis justifient la délégation des modules de l'application et des ressources consommatrices à des serveurs distants en utilisant les plates-formes de services du Cloud Computing. Google offre une de ses principales solutions appelées AppEngine [2].

Actuellement, les architectures des Clouds mobiles sont basées sur le niveau d'abstractions de services du Cloud computing (IaaS, PaaS et SaaS) [3]. Ces architectures répondent à la virtualisation et à la distribution des services déployés. Cependant, l'aspect de la mobilité n'a pas été conçu pour l'utilisation des périphériques mobiles nomades. Notre contribution vise à définir un niveau d'abstraction supplémentaire sur le Cloud, de spécifier une structure qui représente les périphériques mobiles. Les communications émises aux périphériques sont converties selon les protocoles spécifiques en cette structure. Et les réponses sont stockées dans un cache : c'est l'état virtuel du périphérique. Cette représentation agit aussi comme une plate-forme "mobile-friendly" dans le Cloud. En effet, la représentation est construite sur les capacités d'émulation qui offrent un environnement compatible avec le périphérique physique sur lequel la représentation est associée. Nous distinguons 3 types de représentations en fonction de leur association (ou non) avec les périphériques physiques. Le 1er type de représentation s'applique aux simples capteurs et actuateurs. C'est la forme de représentation la plus simple qui agit comme un proxy cache avec une interface commune. Le 2ème type de représentation s'applique aux périphériques mobiles tels qu'une tablette ou un téléphone portable. Cette représentation permet de déporter et conserver l'état des différents capteurs et actuateurs disponibles sur le périphérique mobile. Nous considérons ce type de représentation comme une composition de ressources associées à un périphérique mobile. Ce n'est pas l'image exacte du périphérique mobile. Le 3ème type de représentation n'a aucune association directe avec un périphérique physique. C'est une composition de multiples représentations. Nous la définissons comme une composition de ressources distribuées sur le réseau, qui transforment le périphérique mobile en un "super périphérique" sans les limitations physiques. Cette forme de représentation déportée est un atout pour la gestion d'énergie des périphériques.

Ce document souligne, dans la section II, les différents défis à relever au cours de la virtualisation et comment ils sont traités dans ce travail connexe. Nous décrivons les techniques existantes du Cloud qui sont utiles pour le MCC et présentons les formalismes qui sont liés aux aspects de la virtualisation. Dans la section III, nous

définissons la représentation du périphérique virtuel (VDR : Virtual Device Representation) en utilisant le pi-calcul. Nous abordons l'orchestration et la mise en réseau de ces VDR. La dernière section décrit une étude de cas pour une plate-forme MCC montrant la congruence structurelle entre le système lorsqu'une application mobile est directement en marche sur l'appareil et lorsque cette même application est déportée dans un Cloud mobile.

2. Travaux connexes

Dans nos différents espaces de travail, le rôle des périphériques mobiles a augmenté. Ces espaces de travail ne se limitent plus aux bureaux, ni aux entrepôts. Les périphériques mobiles ont supprimé les limites d'espaces de travail, et par conséquent, les employés peuvent se connecter avec leurs réseaux d'entreprise n'importe où et presque en tout temps. Cependant, pour les IT, l'émergence des technologies dans le domaine de la mobilité génère de nouveaux défis de gestion. Certes, comme l'innovation mobile continue, à savoir l'internet des objets qui permet de rendre des machines communicantes, le concept de machine à machine (M2M) va accélérer davantage les opportunités du mobile [4] et transformer la façon dont les personnes, les entreprises interagissent avec de nombreux aspects de la vie moderne.

Avec une compréhension des meilleures pratiques et l'option de gestion des périphériques mobiles (MDM) [5], les IT peuvent tendre vers la satisfaction de ces défis. Avec une stratégie MDM bien mise en œuvre, les entreprises peuvent appliquer leurs politiques de sécurité sans étouffer la productivité des utilisateurs. Aucune approche uniforme de virtualisation n'a réellement émergé. Seuls les travaux d'Elijah [39] montre la voie de la virtualisation d'application embarquée sans aborder le réseau

A. Cloud, réseau et la virtualisation

La virtualisation est utilisée pour faire abstraction du système d'exploitation (OS ou SE) et des applications, du matériel physique afin de construire un environnement de serveurs plus rentables, agiles et simplifiés. Il existe deux types de virtualisation et de nombreux usages majeurs. Deux sortes de virtualisation sont utilisées pour simuler le matériel de la machine et permettre l'exécution d'un SE invité.

1. Emulation :la machine virtuelle émule ou simule le matériel complet [6].
2. Totale / Native : la machine virtuelle simule suffisamment le matériel pour permettre à un OS invité non modifié de s'exécuter de façon isolé [7] [8].

Nous voyons plusieurs virtualisations de serveurs et de postes de travail. La virtualisation de serveurs définie comme le fait de faire fonctionner plusieurs serveurs virtuels sur un serveur physique, ce dernier étant alors remplacé par son équivalent virtuel. La virtualisation de postes de travail ou client consiste à afficher sur un, des dizaines, centaines voire des milliers de postes physiques, une image virtuelle du poste utilisateur qui est en fait réellement exécutée sur un serveur distant. Selon les statistiques VMware [9], la virtualisation peut offrir 80 % d'utilisation des ressources sur le serveur ou un meilleur ratio de consolidation du serveur. Les efforts actuels visent la formalisation des interactions des services du Cloud [10] et l'orchestration

[11]. Cependant, ces efforts n'abordent pas l'aspect de la virtualisation de tels systèmes du Cloud Computing.

La virtualisation du réseau consiste à combiner des ressources réseaux (matérielles et logicielles) dans une seule unité administrative. L'objectif de la virtualisation du réseau est de fournir aux systèmes et utilisateurs un partage efficace, contrôlé et sécurisé des ressources réseaux. Les ressources de réseaux virtuels sont divisées en 2 catégories : la 1ère est la virtualisation des ressources physiques comme vRouter (routeur) et la 2ème est celle des ressources comme FWA (pare-feu) et LBA (équilibrage de charge). Cette approche de la virtualisation de réseau est appelée NFV (Network Functions Virtualization) [12]. En parallèle à des travaux pragmatiques sur la gestion du réseau, il existe des travaux sur la définition de formalisme dédié. U. Montanari et al ont travaillé sur une extension du langage π-calcul [13] pour les cloud. A. Singh et al ont aussi travaillé sur une extension appelée ω-calcul [14], qui est un formalisme de modélisation et du raisonnement sur les réseaux mobiles sans fil. Dans ce cadre l'usage de logique non temporelle est une limite pour l'étude de la mobilité.

B. Pi-Calcul

Dans le Cloud mobile, les appareils, les services et leurs compositions sont exécutés dans un environnement parallèle et distribué. La représentation de ces périphériques doit suivre un modèle de calcul des systèmes parallèles. Le π-calcul [15] est une algèbre de processus qui offre une syntaxe pour représenter les processus (comme les représentations des périphériques ou des services), la composition parallèle de processus, la communication synchrone entre les processus par le biais de canaux (ou des noms), la création de nouveaux canaux, la réplication de processus et le non-déterminisme. Il fournit des primitives pour décrire et analyser un système distribué, qui utilisent la migration de processus entre pairs, l'interaction des processus via des canaux de communication dynamiques et privés. Les ouvrages de R Milner [16] restent une référence pour une meilleure compréhension de ce langage.

Les 2 concepts importants de la définition π-calcul [16] sont les processus et les canaux. En effet, un processus est une abstraction d'un processus de contrôle qui peut représenter un périphérique sur le Cloud, un service de Cloud Computing comme les fonctions de réseau, ou des business services qui peuvent migrer des périphériques mobiles à la représentation du périphérique mobile sur le Cloud. Un canal est une abstraction de la liaison de communication entre deux processus. Il peut représenter une liaison de service Web entre le périphérique mobile et sa représentation virtuelle en tant que canal de synchronisation. Les canaux peuvent aussi représenter l'interaction de l'envoi et la réception de messages entre un processus local (périphérique mobile) et un autre processus à distance sur le Cloud. Nous ne pouvons détailler dans ce document toutes les notions du Pi calcul, la référence demeure [24].

3. Représentation des périphériques virtuels (VDR)

Dans notre approche, la VDR vise à aborder le paradigme de la virtualisation du MCC. Nous utilisons le sigle VDR comme un label générique pour tous les types de représentation dans le Cloud mobile. Cependant, il y a beaucoup de différence entre

les représentations d'un simple capteur, d'un périphérique mobile, ou composite. Les capteurs pourraient être reliés à leurs VDRs en utilisant certaines normes spécifiques comme 802.15.4 (ZigBee, UWA, ...) [17] qui offrent un taux de transfert acceptable pour synchroniser l'état du capteur avec sa VDR. Pour sa part, le périphérique mobile a besoin d'utiliser des technologies avec un taux de transfert élevé. Cela permet le transfert de données nécessaires pour la déportation de l'application vers le Cloud mobile. La VDR composite n'admet pas de connexion directe au périphérique physique, elle communique avec les périphériques physiques par le biais de leur représentation virtuelle en utilisant un bus d'événement (Event Bus). La Figure 1 donne un aperçu des trois différents types de VDR identifiées. En raison de leurs différences, les trois types de VDRs ont une définition formelle spécifique définie par les équations (3), (6) et (9). Ils ont également des rôles spécifiques dans le Cloud mobile. VDR capteur (SVDR): elle représente un capteur physique ou un actuateur au sein du Cloud mobile, comme une LED ou un capteur de pression ; VDR du périphérique mobile (DVDR) : elle représente un périphérique mobile physique au sein du Cloud mobile comme un smartphone ; VDR Composite (CVDR): elle représente une composition de SVDR, de DVDR et des ressources du Cloud mobile comme l'agrégation de toutes les caméras contenues dans une salle de conférence.

Dans cette section, nous présentons les différents aspects tournant autour de la VDR en, donnant notre définition formelle en Pi-calcul d'ordre supérieur (HOπC) [18], soulignant le mécanisme d'orchestration pour la VDR et de la gestion du réseau.

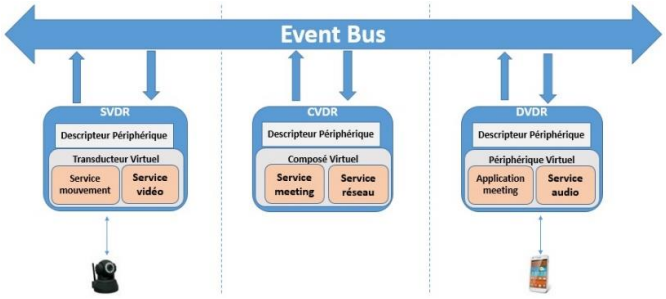

Figure 1. Connections des VDRs

La VDR est définie comme la composition des ressources (CPU, RAM, …), les périphériques mobiles et des capteurs. Il est un composant logiciel composite qui fournit une émulation du comportement du matériel physique qu'il représente. Une VDR est une instance d'une VM légère utilisée dans le Cloud computing, hébergeant un SE mobile et exposant des services de gestion qui émulent un écran d'affichage et / ou un clavier. Elle peut exécuter plusieurs types d'applications mobiles (connus sous le nom apps) et admet une connexion réseau.

A. Spécification formelle

La VDR fonctionne selon une architecture orientée événements. Toutes les interactions sont initiées par un message envoyé à partir d'un bus ou par un appel de service. Le message survient à la suite de l'activité de détection du matériel. Nous définissons un vecteur d'événements représentant toutes les interfaces d'une VDR. Ce vecteur d'événements contient des canaux qui sont utilisés pour échanger des messages. Le bus d'événements (Event Bus) est au cœur de notre modèle formel.

Comme illustré sur la Figure 2, le bus d'événements est le composant qui permet la propagation des événements de détection dans un système distribué. En effet, les architectures des périphériques mobiles [19] [20] [21] ne sont pas conçues pour permettre une intégration directe des périphériques distants. Pour permettre ce type d'intégration, nous avons ajouté deux composants dans la couche mobile, appelés le service capteur et le pilote dynamique. Ces composants visent à propager des évènements de capteurs embarqués dans les appareils mobiles, de récupérer les événements de détections externes, et d'intégrer ces événements au niveau du noyau dans le système d'exploitation mobile (SE). Le vecteur d'événements n'admet pas une définition formelle statique. Il est défini comme un vecteur qui contient autant de canaux que des capteurs du système tel quel illustré par l'exemple équation (1).

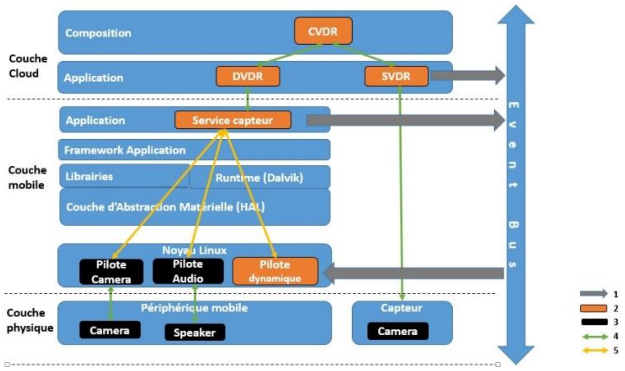

Figure 2. Rôle de Event Bus (1 : propagation de l'événement, 2 : Composants standards, 3 : Composants proposés, 4 : Communication via la couche application, 5 : Communication directe)

$$\overrightarrow{evt} \stackrel{\text{def}}{=} [id, c, camera_1, micro_1, micro_2, nfc_1, keyboard_1, \dots] \qquad (1)$$

Le vecteur d'événements contient par défaut deux éléments indexés par id et c. L'élément id est utilisé pour récupérer l'identifiant de la VDR. L'élément c est utilisé pour ajouter une nouvelle VDR à une composition existante. L'équation (12) illustre cette opération d'ajout ou de concaténation. Le vecteur d'événements $\overrightarrow{ev}$ est utilisé uniquement pour les interactions entre les VDRs. Les interactions entre les DVDRs d'une part, des SVDRs et l'appareil physique d'autre part utilisent un service basé sur un canal appelé ws_i qui représente l'échange des services Web.

$$VDR(\overrightarrow{ws}) \stackrel{\text{def}}{=} ((\lambda\, \overrightarrow{ev}\, ws_1)SVDR + (\lambda\, \overrightarrow{ev}\, ws_2)DVDR + (\lambda\, \overrightarrow{ev}\, ws_3)CVDR + \emptyset) \qquad (2)$$

Le terme $VDR(\overrightarrow{ws})$ admet comme paramètre un vecteur $\overrightarrow{ws}$ de canaux de services Web. Ce vecteur est partagé entre le système du Cloud mobile et les VDRs. Les canaux du $\overrightarrow{ws}$ permettent les échanges avec les périphériques physiques. Le terme VDR crée un nouveau vecteur, appelé $\overrightarrow{ev}$ qui contient les canaux des interfaces VDRs. Nous bénéficions de l'utilisation de l'abstraction dans la définition de la VDR où le processus $(\lambda\ \overrightarrow{ev}\ ws_1)SVDR$ est écrit de façon naturelle en $SVDR(\overrightarrow{ev}, ws)$, et ainsi de suite pour les deux autres DVDRs. La VDR spécifique est activée si l'élément correspondant dans le vecteur de $\overrightarrow{ws}$ est un canal valide et non un processus vide $\emptyset$.

$$TwoSensors(ws_1, ws_2) \overset{\text{def}}{=} VDR(ws_1\hat{}\emptyset) \frown VDR(ws_2\hat{}\emptyset) \quad (3)$$

L'utilisateur final se doit de définir d'une façon plus concrète le comportement des périphériques. Pour illustrer une telle définition, nous donnons dans l'équation (4) un exemple de capteur NFC (Near Field Communication) [23] qui envoie la valeur « $sens$ » sur le canal $nfc1$.

$$Cnfc(\overrightarrow{ev}, ws) \overset{\text{def}}{=} ws(sens).\tau.\overline{ev_{nfc_1}}\langle sens\rangle \quad (4)$$

La SVDR est activé suite à l'événement de connexion au capteur physique. Une fois connecté, le capteur physique envoie la donnée d'identification (id) au SVDR à travers le canal ws. Cette donnée est stockée à l'intérieur de la SVDR à travers le terme $DevId$ défini dans l'équation (7), qui redonne la donnée d'identification sur demande par le canal d'événements.

$$SVDR(\overrightarrow{ev}, ws) \overset{\text{def}}{=} ws(id).\tau.(DevId(ev_{id}, id)|VirtualSensor(\overrightarrow{ev}, ws)) \quad (5)$$

Comme illustré dans l'équation (5), le terme $SVDR$ utilise le terme $VirtualSensor$ défini dans l'équation (6) pour envoyer les données perçues par le capteur physique en utilisant le canal d'événements. 2 comportements possibles peuvent être adoptés par le terme $VirtualSensor$ comme illustré dans l'équation (6) : si une commande Stop de l'équation (17) est reçue, le processus prendra fin ; sinon l'action de dispatching est exécutée. La composition parallèle du terme $SVDR$ permet à l'administrateur de récupérer l'identifiant de la VDR en utilisant le canal ev_i. Cette composition n'influence pas l'exécution du capteur virtuel.

$$VirtualSensor(\overrightarrow{ev}, ws) \overset{\text{def}}{=} \quad (6)$$
$$ws(C).([C = Stop]\ Stop + C(\overrightarrow{ev}, ws).VirtualSensor(\overrightarrow{ev}, ws))$$

$$DevId(req, id) \overset{\text{def}}{=} req(cb).\overline{cb}\langle id\rangle.DevId(req, id) \quad (7)$$

La spécification de la DVDR respecte les mêmes fondamentaux que la SVDR. Comme illustré dans l'équation (8), il utilise le terme $DevId$ pour stocker et redonner l'identifiant du périphérique. Il utilise le terme appelé $VirtualDevice$ pour gérer le comportement du périphérique virtuel. Cependant, la DVDR a la capacité d'exécuter des applications qui sont lies aux évènements du capteur à la place de la SVDR.

$$DVDR(\overrightarrow{ev}, ws) \overset{\text{def}}{=} ws(id).\tau.\left(DevId(ev_{id}, id)\middle|DevId(ev_{id}, id)\right) \quad (8)$$

Nous avons besoin de dissocier les événements de détection envoyés par les capteurs de périphériques embarqués et les demandes d'application déportée. Pour ce faire, nous définissons un type appelé App de l'équation (9) qui encapsule l'application déportée et agit comme un conteneur d'application.

$$App\big(BackEndProc(ws)\big) \stackrel{\text{def}}{\cong} BackEndProc(ws) \qquad (9)$$

Nous définissons dans l'équation (10) le terme $VirtualDevice$ qui exécute l'application déportée si nécessaire, sinon, il mandate les données de détection.

$$VirtualDevice(\overrightarrow{ev}, ws) \stackrel{\text{def}}{\cong} ws(C).Com\,.VirtualDevice(\overrightarrow{ev}, ws) \qquad (10)$$

$$Com \stackrel{\text{def}}{\cong} \big([C = Stop]Stop + \big[C = App\big(P(x)\big)\big].P(x) + C(\overrightarrow{ev}, ws)\big) \qquad (10.1)$$

La principale préoccupation de ce terme est de faire la distinction entre l'action de déportation représentée par $App\big(P(x)\big)$, l'action d'arrêt et les actions de détection. Pour ce faire, nous comparons la structure du nom C reçu via le canal de ws. Le $VirtualDevice$ exécute le paramètre d'ordre supérieur $P(x)$ au sein de la DVDR sur l'action de déportation, appelle le terme $Stop()$ sur l'action d'arrêt. La CVDR de l'équation (11) ne possède aucune association directe avec un périphérique physique, ses interactions passent à travers une SVDR ou une DVDR. Le terme $CVDR$ est défini comme une agrégation de la SVDR et de la DVDR, qui partage le même vecteur d'événements.

$$CVDR(\overrightarrow{ev}, C) \stackrel{\text{def}}{\cong} (\nu\, id)DevId(ev_{id}, id)|CompositeDevice(\overrightarrow{ev}) \frown C \qquad (11)$$

Un nouvel identifiant est créé du terme $CVDR$ et remis à la demande en utilisant le canal d'événement ev_{id} du terme $DevId$.

$$CompositeDevice(\overrightarrow{ev}) \stackrel{\text{def}}{\cong} ev_c(\vec{e}).\Big(CompositeDevice(\overrightarrow{ev})\Big\{{}^{\overrightarrow{ev}\hat{}\vec{e}}/_{\overrightarrow{ev}}\Big\}\Big) \qquad (12)$$

Le terme $CompositeDevic$e défini dans l'équation (12) est utilisé pour agréger les canaux d'événements. Le vecteur $\overrightarrow{ev}$ est un canal d'événements associé au terme $CompositeDevice$ actuel tandis que le vecteur $\vec{e}$ est le canal d'événements associé à la VDR pour l'appartenance à cette composition. Cette agrégation est basée sur deux opérateurs : le premier est l'opérateur $\{{}^{x}/_{y}\}$ de changement de nom de CCS [24]. Le deuxième est l'opérateur ^ de concaténation. Le résultat de cette combinaison est l'utilisation de la concaténation de deux vecteurs d'événements $\overrightarrow{ev}\hat{}\vec{e}$ à la place du vecteur d'événements qui a été associé au terme $CompositeDevice$.

B. Orchestration

L'orchestration du Cloud mobile vise à automatiser la configuration, la coordination et la gestion des VDRs et leurs interactions dans un tel environnement. Le procédé consiste à automatiser les flux de travail nécessaires à la composition des VDRs et à la déportation des applications mobiles. Dans notre approche, l'orchestrateur est composé de trois principaux composants, comme l'illustre l'équation (13) ; Nous définissons ces trois composants comme des tâches communes

d'orchestration : Configuration où l'orchestrateur du Cloud gère le stockage, le calcul et la mise en réseau ; Dans cet article, nous ne nous concentrons pas sur l'algorithme d'allocation des ressources (de calcul et de stockage); cet aspect sera mentionné dans notre future recherche. Une spécification de haut niveau du mécanisme de gestion du réseau est présentée dans la section 3.D. Approvisionnement où l'orchestrateur du Cloud gère les VDR en fournissant l'exécution, suspension et l'arrêt complet des opérations. Sécurité où l'orchestrateur du Cloud gère le suivi et le reporting. Nous avons décrit les détails de cet aspect également sur un document distinct [24] où nous décrivons notre implémentation et des algorithmes détaillés.

$$Orchestrator(\overrightarrow{api}) \stackrel{\text{def}}{=} Configuration(\overrightarrow{api}) \mid \text{Provisioning}(\overrightarrow{api}) \mid (\nu\ \overrightarrow{data})Monitoring(\overrightarrow{api}, \overrightarrow{data}) \quad (13)$$

Dans le terme $Configuration$ de l'équation (14), nous illustrons la configuration de l'api qui est utilisée pour l'allocation, la désaffectation (libération) et la suspension de l'exécution des ressources. L'api est un vecteur dans un espace à deux dimensions. L'exposant indique le module cible (exemple : $api^{\boldsymbol{c}}$ où $\boldsymbol{c}$ signifie $\boldsymbol{C}onfiguration$). L'indices indiquent le service appelé dans le module (exemple : api_a où a réservé pour l'allocation). L'administrateur système utilisera le vecteur $\overrightarrow{api}$ pour configurer l'environnement de déploiement. Nous notons ici que le système d'allocation ou de libération des ressources n'est pas pris en compte. A ce niveau, ces opérations ne sont pas observables. Cependant, nous associons un nouveau nom à chaque demande d'allocation reçue de l'administrateur sur les $\boldsymbol{api_a^c}$.

$$Configuration(\overrightarrow{api}) \stackrel{\text{def}}{=} (api_a^c(allocate).\tau.(\nu\ res)\overline{allocate}\langle res\rangle \mid api_f^c(free).\tau \mid api_s^c(suspend).\tau)\ .Configuration(\overrightarrow{api}) \quad (14)$$

Le terme Provisioning de l'équation (15) utilise également une api de demande du module de configuration pour l'allocation des ressources. Une fois attribuée, elle reçoit le terme, exécute le paramètre d'ordre supérieur pour permettre la création de la VDR. Le paramètre est une instance du terme Run défini dans l'équation (16). Nous utilisons l'abstraction des ressources d'information res, retournée par le terme $Configuration$, pour communiquer cette information au terme Run qui est préconfiguré avec les deux paramètres avant sa réception à travers le canal api_r^p.

$$\text{Provisioning}(\overrightarrow{api}) \stackrel{\text{def}}{=}$$

$$\Big(\big(\ api_r^p(Run).(\nu\ allocate)\overline{api_a^c}\langle allocate\rangle \mid allocate(res).(\lambda\ res)Run(\ ,\ )\big) \mid api_s^p(suspend).\overline{api_s^c}\langle suspend\rangle \mid (\ api_t^p(terminate).terminate(ws)\ .\overline{ws}\langle Stop\rangle.\overline{api_f^c}\langle terminate\rangle)\Big)\ .\text{Provisioning}(\overrightarrow{api}) \quad (15)$$

La suspension est déléguée au terme $Configuration$ où il est représenté comme une action non observable τ. Le terme Provisioning envoie le terme $Stop$ terme à la VDR pour terminer son exécution. Nous utilisons pour cela le canal ws envoyé à travers le

canal $terminate$.Le terme Run défini dans l'équation (16) compose un vecteur selon le type de la VDR et en fonction du type demandé en utilisant le vecteur $\overrightarrow{type}$.

$$Run(ws\ ,\overrightarrow{type}) \stackrel{\text{def}}{=} [type_v = type_c](v\ id)\overline{ws}\langle id\rangle$$
$$\left|\tau.(v\ \overrightarrow{ev})\left(\begin{array}{c}[type_v = type_s]\ VDR(ws\hat{}\emptyset\hat{}\emptyset)\\ |\ [type_v = type_d]\ VDR(\emptyset\hat{}ws\hat{}\emptyset)|[type_v = type_c]\ VDR(\emptyset\hat{}\emptyset\hat{}ws)\end{array}\right)\right. \quad (16)$$

Après la création de la VDR, il crée et envoie un nouvel identifiant en utilisant le canal ws. Cet identifiant permet d'utiliser la nouvelle VDR créée. Les émissions sur le canal ($\overline{ws}\langle id\rangle$) avec le nouvel identifiant créé font la paire avec les équations (5) et (8).

$$Stop(\ ) \stackrel{\text{def}}{=} \emptyset \quad (17)$$

Pour conserver le sens de notre définition, nous n'avons pas intégré les communications entre les VDRs et le module de monitoring défini par le terme $Monitoring$. Facilement, nous pouvons imaginer qu'après chaque communication sur le canal du vecteur d'événements $\overrightarrow{ev}$, une information doit être transmise au module de monitoring en utilisant le canal api^m_{put}. Cette information est stockée dans le vecteur de données $\overrightarrow{data}$ à travers l'appel récursif du terme $Monitoring$ défini par l'équation (18).

$$Monitoring(\overrightarrow{api},\overrightarrow{data}) \stackrel{\text{def}}{=}$$
$$(api^m_{put}(datum).\tau.(v\ id)\overline{api^m_{ret}}\langle id\rangle\ |api^m_{ged}(id).api^m_{res}(data_{id}))$$
$$.Monitoring(\overrightarrow{api},\overrightarrow{data}\hat{}datum) \quad (18)$$

D. Networking

Dans notre approche du Cloud mobile, plusieurs entités peuvent utiliser la même infrastructure physique. L'infrastructure partagée permet l'indépendance des VDRs vis-à-vis de l'hôte physique sur lequel il est situé. Le modèle de réseau virtuel proposé, permet au module d'approvisionnement (équation 15) de gérer le composant du réseau virtuel comme une VDR et de cacher la complexité à l'utilisateur. Pour notre modélisation du réseau, nous définissons la structure du paquet qui transite sur les infrastructures du réseau. Le vecteur $\overrightarrow{ethernet}$ dans l'équation (19) représente la trame L2 où les noms $ethernet_{dst}$ et $ethernet_{src}$ sont les canaux correspondant au ws_x utilisé par les VDRs dans l'équation (2). $ethernet_{ip}$ contient les informations nécessaires pour le terme $vRouter$ et aussi les messages comme $ip_{payload}$. Les noms qui appartiennent à des vecteurs dans les équations (19) et (20) sont des abréviations des champs d'entêtes de paquets tels que décrits dans IETF RFC 791.

$$\overrightarrow{ethernet} \stackrel{\text{def}}{=} [dst, src, tag, type, \overrightarrow{ip}, check] \quad (19)$$
$$\overrightarrow{ip} \stackrel{\text{def}}{=} \left[\begin{array}{c}version, ihl, tos, len, id, flag, frag, ttl, proto,\\ check, src, dst, opt, payload\end{array}\right] \quad (20)$$

Étant donné que notre objectif n'est pas d'insister sur les protocoles de réseau, mais de souligner les communications entre les composants virtuels, nous avons fait

abstraction de tout comportement réseau qui n'est pas directement lié à la virtualisation. Nous les avons nommé comme des opérations non observables τ.

$$Switch\left(cntl, \overrightarrow{adr}\right) \stackrel{\text{def}}{\equiv} Control\left(vSwitch\left(cntl, \overrightarrow{adr}\right), cntl, \overrightarrow{adr}\right) \quad (21)$$
$$| \; adr_i\left(\overrightarrow{ethernet}\right).\tau.\overline{ethernet_{dst}}\langle ethernet_{ip_{payload}}\rangle.vSwitch\left(cntl, \overrightarrow{adr}\right)$$

Le terme $vSwitch$ défini dans l'équation (21) représente la virtualisation du commutateur L2. Il est modélisé comme une congruence entre le terme $Control$ défini dans l'équation (22), qui gère les connexions des VDRs et un pont réseau de L2. Afin de définir les termes des équations (22) et (23), nous utilisons la notation et la définition numérale introduites par R. Milner [22] section 3.3 par exemple : $\underline{n}(x,z) \stackrel{\text{def}}{\equiv} (\overline{x}.)^n \, \overline{z}$. Milner définit le successeur sous la forme :

$$(\nu \, xz)\left(\underline{n}(x,z)\middle| Succ(xz, yw)\right) \approx \underline{n+1}(y,w)$$

Les numérales sont utilisées comme des indices dans le but de gérer dynamiquement le contrôle de la connexion / déconnexion des périphériques.

$$Control\left(Target, cntl, \overrightarrow{adr}\right) \stackrel{\text{def}}{\equiv} cntl_{connect}(link).Target \; | \quad (22)$$
$$cntl_{disconnect}(link).(\nu \; i \; z)\left(Disconnect\left(Target, \overrightarrow{adr}, (\nu \; \vec{p}), link, \underline{0}(i,z)\right)\right)$$

$$Disconnect\left(Target, \overrightarrow{old}, \overrightarrow{adr}, port, \underline{n}(i,z)\right)$$
$$\stackrel{\text{def}}{\equiv} \left[\underline{0+n}(i,z) \approx \underline{0+\left\|\overrightarrow{old}\right\|}(y,z)\right]\left(\lambda \; \overrightarrow{adr}\right)Target$$
$$| \; [old_i = port] \; Disconnect\left(Target, \overrightarrow{old}, \overrightarrow{adr}, port, (\nu \; y)\underline{n+1}\,(y,z)\right)$$
$$| \; Disconnect\left(Target, \overrightarrow{old}, \overrightarrow{new}\hat{}port, port, (\nu \; y)\underline{n+1(y,z)}\right) \quad (23)$$

Le terme $Control$ prend trois paramètres. Le premier est le paramètre d'ordre supérieur appelé $Target$, qui est utilisé pour transmettre les termes $vSwitch$ et $vRouter$. Le second est appelé $cntl$, il est utilisé comme un canal pour contrôler les connexions des VDRs. Le troisième paramètre est le vecteur contenant les canaux des VDRs connectés. Le paramètre $Target$ est paramétré également dans le terme $Disconnect$ défini dans l'équation (23), nous utilisons l'abstraction λ pour remplacer le vecteur d'adresse $\overrightarrow{adr}$ qui reste un nom libre dans $Target$ quand un périphérique est déconnecté.

$$vRouter\left(ipAdr, cntl, \overrightarrow{adr}\right) \stackrel{\text{def}}{\equiv} Control\left(vRouter\left(ipAdr, cntl, \overrightarrow{adr}\right), cntl, \overrightarrow{adr}\right)$$
$$| \; adr_i\left(\overrightarrow{ethernet}\right).\tau.\left(\left[ipAdr = ethernet_{ip_{dest}}\right]\overline{ethernet_{dst}}\langle\overrightarrow{ethernet}\rangle\right.$$
$$\left. + \overline{ethernet_{ip_{dest}}}\langle\overrightarrow{ethernet}\rangle\right).vRouter\left(cntl, \overrightarrow{adr}\hat{}link\right) \quad (24)$$

Le terme $vRouter$ défini dans l'équation (24) représente la virtualisation du routage L3. Il est modélisé comme une concurrence entre un processus Control de l'équation (22) qui gère les commutateurs virtuels ou les connexions VDRs et un pont réseau de L3. Dans ce modèle, nous n'illustrons pas certaines fonctionnalités comme le routage IP afin de rester dans notre contexte de définition. La gestion de l'infrastructure réseau est exposée dans la partie de l'API provisioning. Pour ce faire, nous illustrons dans l'équation (25) une extension du terme $Provisioning$ défini initialement dans l'équation (15). Nous ajoutons à cette extension la possibilité de

connecter et de déconnecter des périphériques. En outre, cette extension offre la possibilité de créer un nouveau commutateur L2.

$$\text{Provisioning}(\overrightarrow{api}) \stackrel{\text{def}}{=} \left(\begin{array}{l} \ldots \\ \left| \begin{array}{l} api^{p}_{vsCreate}(ret).(\nu\ cntl) \\ \quad \left(\tau.\left(\nu\ \overrightarrow{adr}\right) vSwitch\ \left(cntl, \overrightarrow{adr}\right) \right) \\ \quad\quad \mid \overline{ret}\langle cntl \rangle \end{array} \right. \\ \left| api^{p}_{vsConnect}(cntl, adr).\tau.cntl_{connect}(adr) \right. \\ \left| api^{p}_{vsDisconnect}(cntl, adr).\tau.cntl_{disconnect}(adr) \right. \\ \ldots \end{array} \right) . Provisioning(\overrightarrow{api}) \quad (25)$$

Dans l'équation (14), nous décrivons le commutateur lié à l'API Provisioning, le canal $api^{p}_{vsCreate}$ est utilisé pour créer le commutateur virtuel, ensuite retourner le canal de contrôle $cntl$ à l'initiateur de la demande. L'API Provisioning du routeur est comparable à l'API du Switch, la seule différence est que les canaux api^{p}_{vs*} sont définis comme api^{p}_{vr*} et $api^{p}_{vrCreate}$. Ils sont utilisés pour créer un routeur virtuel. Pour conserver le sens de notre définition, nous avons omis cette partie. Notre modèle formel vise à définir une nouvelle architecture de l'approche basée sur la Cloudlet.

4. Etude de Cas

Notre étude de cas vise à montrer la congruence structurelle entre un Backend app déportée dans une VDR et le même Backend app fonctionnant dans le périphérique mobile. Notre objectif est d'illustrer le fait qu'une Backend application (voir l'équation (9)) qui fonctionne dans une VDR est identique de façon structurelle, à une Backend application qui fonctionne dans un périphérique mobile. Le but est noter les différences apportées à une application embarquée avant et après migration dans une cloudlet. Ce résultat est obtenu par réduction des 2 systèmes à un système identique.

A. Périphérique mobile

Nous définissons d'abord les termes $FrontEnd$ et $BackEnd$. Le terme $FrontEnd$ représente l'application cliente et le terme $BackEnd$ représente le service utilisé dans notre étude de cas. Ces termes sont les composants des périphériques mobiles définis dans les équations (28) et (29). Le terme $FrontEnd$, défini dans l'équation (26), est un modèle d'une "vue web" qui envoie des messages au $BackEnd$ utilisant le canal ws, une fois la réponse reçue par le $BackEnd$ (serveur), le $FrontEnd$ exécute une autre itération de façon récursive. $FrontEnd$ a également le canal $touch$ comme paramètre pour communiquer avec l'utilisateur défini dans l'équation (31).

$$FrontEnd(touch, ws) \stackrel{\text{def}}{=} (\nu\ cb)\ touch(event).\tau.\overline{ws}\langle event, cb \rangle \mid cb(res).FrontEnd(ws) \quad (26)$$

Le terme $BackEnd$, défini dans l'équation (27), réagit au message envoyé par le $FrontEnd$. Si l'abstraction du canal $intra$ est liée au même canal que le paramètre

ws, le $BackEnd$ app est exécuté localement sur le périphérique mobile. Sinon, le $BackEnd$ envoie un message contenant une copie de lui pour la VDR correspondante et termine l'exécution locale. L'exécution continue dans la VDR après la déportation.

$$BackEnd(ws) \stackrel{\text{def}}{=} (\lambda\ intra)\ ws(event, cb).\tau \cdot \begin{pmatrix} [intra = ws].\overline{intra}\langle\ \ \rangle \\ + \overline{ws}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset \end{pmatrix} \Big| intra(\ \ ).\tau.(\nu\ res)\overline{cb}\langle res\rangle \quad (27)$$

Nous définissons 2 compositions parallèles en tant que modèles pour les périphériques mobiles. Le 1er périphérique mobile est défini dans l'équation (28) comme l'exécution en parallèle d'un $FrontEnd$ et d'un $BackEnd$ en local. Le 2ème périphérique mobile est défini dans l'équation (29) comme l'exécution en parallèle d'un $FrontEnd$ et d'un $BackEnd$ qui est configuré pour être déporté dans la VDR.

$$Devicelocal(ws, touch) \stackrel{\text{def}}{=} FrontEnd(touch, ws) | (\lambda\ ws)BackEnd(ws) \quad (28)$$

$$DeviceRemote(ws, touch) \stackrel{\text{def}}{=}$$

$$(\nu\ local)(FrontEnd(touch, local) | (\lambda\ local)BackEnd(ws)) \quad (29)$$

Pour garder la clarté de notre spécification, nous omettons les détails de la définition du terme $Admin$, nous définissons simplement sa signature dans l'équation (29). Il est important de noter que ce terme envoie tous les messages nécessaires en utilisant le vecteur $\overrightarrow{api}$. Il concerne l'infrastructure du networking et des VDRs.

$$Admin(ws, \overrightarrow{api}) \stackrel{\text{def}}{=} \cdots \quad (30)$$

Le terme $user$ défini dans l'équation (31) représente un utilisateur de l'appareil en exécutant une action unique par l'envoi d'un événement au $FrontEnd$ à travers le canal $touch$ qui représente l'écran tactile de l'appareil. Nous avons défini une action simple pour l'utilisateur parce que nous voulons un système qui peut-être manuellement réduit par une personne quelconque dans un intervalle de temps raisonnable.

$$User(touch) \stackrel{\text{def}}{=} (\nu\ event)\overline{touch}\langle event\rangle \quad (31)$$

B. Systèmes

Pour être en mesure de vérifier la congruence structurelle, nous définissons deux systèmes comme une composition parallèle de l'utilisateur mobile, du périphérique mobile, de l'administrateur et de l'orchestrateur. Le terme $SystemMig$ défini dans l'équation (32) représente le système qui conduira à une déportation de $BackEnd$ après quelques réductions.

$$SystemMig \stackrel{\text{def}}{=} (\nu\ ws)\begin{pmatrix} (\nu\ touch)\begin{pmatrix} user(touch) \\ |DeviceRemote(ws, touch) \end{pmatrix} \\ \Big|(\nu\ \overrightarrow{api})\begin{pmatrix} Admin(ws, \overrightarrow{api}) \\ |Orchestrator(\overrightarrow{api}) \end{pmatrix} \end{pmatrix} \quad (32)$$

Le terme $SystemLocal$ défini dans l'équation (33) représente le système qui permettra à l'élévation d'un $BackEnd$ exécuté localement après certaines réductions.

$$SystemLocal \triangleq^{\text{def}} (\nu\ ws)\begin{pmatrix}(\nu\ touch)\begin{pmatrix} user(touch) \\ |Devicelocal(ws, touch)\end{pmatrix} \\ \Big|(\nu\ \overrightarrow{api})\begin{pmatrix} Admin(ws, \overrightarrow{api}) \\ |Orchestrator(\overrightarrow{api})\end{pmatrix}\end{pmatrix} \quad (33)$$

C. Congruence Structurelle

Nous avons effectué quelques étapes de calculs pour atteindre pleinement un système stable à partir de $SystemMig$. Le calcul commence par l'envoi d'un événement (event) tactile via le canal touch ($\overline{touch}\langle event\rangle$) depuis un utilisateur $User$ (31) et se termine par la réception du signal d'arrêt $\overline{ws}\langle Stop\rangle$ sur le périphérique virtuel $VirtualDevice$ (11). Nous appelons $SystemMig'$cet état stable atteint après ces réductions où

$$SystemMig \xrightarrow{\overline{touch}\langle event\rangle,\dots} SystemMig'$$

Nous avons appliqué la même opération au terme $SystemLocal$. Toutefois, la réduction de ce système est plus simple, liée au fait de n'avoir aucune réduction de déportation. Ainsi, nous obtenons $SystemLocal'$:

$$SystemLocal \xrightarrow{\overline{touch}\langle event\rangle,\dots} SystemLocal'$$

La congruence structurelle définie [22], est une relation de congruence sur un processus respectant les lois suivantes :

- Agents sont identiques s'ils ne diffèrent que par un changement de noms
- $(\mathcal{N}/\equiv, +, \emptyset)$est un monoïde symétrique
- $(\mathcal{P}/\equiv, |, \emptyset)$est un monoïde symétrique
- $!P \equiv P \mid !P$
- $(\nu\ x)\emptyset \equiv \emptyset, (\nu\ x)(\nu\ y)P \equiv (\nu\ y)(\nu\ x)P$
- si $x \notin fn(P)$ alors $(\nu\ x)(P|Q) \equiv\ P \mid (\nu\ x)Q$

Où $fn(P)$ est définie comme l'ensemble des noms libres du processus P.

Après la réduction des termes $SystemMig$ et $SystemLocal$, nous avons observé que seuls certains noms liés et les actions non observables composaient la différence entre les deux systèmes réduits. Nous pouvons donc prétendre à l'équivalence structurelle des 2 systèmes :

$$SystemMig' \equiv SystemLocal'$$

Par définition, la congruence structurelle est commutative et associative. Donc :

$$\begin{aligned} &\text{Étant donné} && SystemMig \equiv SystemMig' \\ &\text{et} && SystemLocal \equiv SystemLocal' \\ &\text{et} && SystemMig' \equiv SystemLocal' \\ &\text{alors} && SystemMig \equiv SystemLocal \end{aligned} \quad (34)$$

5. Conclusion et Perspectives

Dans ce document, nous présentons notre vision d'un système MCC. Nous fournissons certaines équations de notre définition formelle du MCC pour exprimer notre architecture d'un tel système. Toutes les équations présentées dans le document, concernent uniquement les interactions de la communication entre les composants du système MCC. Nous présentons également certains aspects de notre architecture dédiée à la réalisation d'une solution MCC. Notre étude de cas présente deux systèmes où le premier implique la déportation d'une application et le second implique l'exécution locale de la même application. Nous démontrons dans cette étude, la congruence structurelle entre la déportation et l'exécution locale d'une application mobile. Nous considérons cela comme une preuve de la transparence de la déportation dans notre système MCC. Ainsi, ce travail est une base pour la définition de métriques.

Dans nos travaux futurs, il est question d'aborder deux aspects de la MCC. Le premier aspect concerne une définition formelle d'une métrique pour la définition d'une unité de mesure de la migration des applications. Le deuxième aspect est la définition de la collecte de données et d'un algorithme permettant de calculer le coût de la déportation d'une application. Aussi, comment analyser les données recueillies afin de définir quand la déportation peut-être optimale dans le contexte MCC.

En parallèle à notre travail sur la Cloudlet, nous étudions les possibilités des applications d'un tel système en particulier pour stimuler l'intelligence des bâtiments dits intelligents (Smart-buildings) et leurs apports sur la composition des périphériques au sein de l'internet des objets (IoT).

Références

[5] [1] C. Kyung-Yong, J. Yoo and K. Kim J., "Recent trends on mobile computing and future networks.," Personal and Ubiquitous Computing, vol. 18, no. 3, pp. 489-491, 2014.

[6] [2] D. Sanderson, Programming google app engine: build and run scalable web apps on google's infrastructure., O'Reilly Media, Inc., 2009.

[7] [3] M. Armbrust, "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010.

[4] W. Geng, S. Talwar, K. Johnsson, N. Himayat and K. D. Johnson, "M2M: From mobile to embedded internet.," IEEE Communications Magazine, vol. 49, no. 4 , pp. 36-43, 2011.

[8] L. Liu, R. Moulic and D. Shea, "Cloud service portal for mobile device management," IEEE 7th International Conference on e-Business Engineering (ICEBE), pp. 474--478, 2010.

[9] D. Bartholomew, "Qemu a multihost multitarget emulator," Linux Journal, no. 145, p. 3, 2006.

[10] E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman and E. Y. Wang, "Bringing virtualization to the x86 architecture with the original vmware workstation," ACM Transactions on Computer Systems (TOCS), vol. 30, no. 4, p. 12, 2012.

[11] A. Velte and T. Velte, Microsoft virtualization with Hyper-V, McGraw-Hill, Inc., 2009.

[12] B. Walters, "VMware virtual platform," Linux journal, vol. 63, p. 6, 1999.

[13] J. Jiulei, L. Jiajin, H. Feng, W. Yan and S. Jie, "Formalizing Cloud Service Interactions," Journal of Convergence Information Technology, vol. 7, no. 13, 2012.

[14] C. Mahmoudi, Orchestration d'agents mobiles en communauté, Universite Paris-Est Creteil, 2014.

[15] N. M. K. Chowdhuryr and R. Boutaba, "A survey of network virtualization," Computer Networks, vol. 54, no. 5, pp. 862-876, 2010.

[16] M. Ugo and S. Matteo, Network conscious pi-calculus, Pisa: Universita di Pisa, 2012.

[17] A. Singh, C. Ramakrishnan and S. A. Smolka, "A process calculus for mobile ad hoc networks," Science of Computer Programming, vol. 75, no. 6, pp. 440-469, 2010.

[18] D. Sangiorgi and D. Walker, The pi-calculus: a Theory of Mobile Processes, Cambridge university press, 2003.

[19] R. Milner, Communicating and mobile systems: the pi calculus, Cambridge university press, 1999.

[20] R. Milner, P. Joachim and W. David, "A calculus of mobile processes," Information and computation , vol. 100, no. 1, pp. 1-40, 1992.

[21] R. {Napier and M. Kumar, iOS 7 Programming Pushing the Limits: Develop Advance Applications for Apple iPhone, iPad, and iPod Touch, John Wiley & Sons, 2014.

[22] J. Annuzzi Jr, L. Darcey and S. Conder, Advanced Android Application Development, Addison-Wesley Professional, 2014.

[23] J. Wingfield, Developing a Windows Phone Application, Cardiff Metropolitan University, 2014.

[24] R. Milner, The polyadic π-calculus: a tutorial, Berlin Heidelberg: Springer, 1993.

[25] G. Milette and A. Stroud, Professional Android sensor programming, John Wiley & Sons, 2012.

[26] R. Milner, A calculus of communicating systems, Springer, 1980.

[27] C. :. M. F. Mahmoudi, "Monitoring Architecture for Cloudlet Based Mobile Cloud Computing,".

[28] T. Sridhar, L. Kreeger, D. Dutt, C. Wright, M. Bursell, M. Mahalingam, P. Agarwal and K. Duda, Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, IETF, 2014.

[29] H. T. Dinh, C. Lee, D. Niyato and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches.," Wireless communications and mobile computing, vol. 13, no. 18, pp. 1587-1611, 2013.

[30] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI: making smartphones last longer with code offload," Proceedings of the 8th international conference on Mobile systems, applications, and services, pp. 49-62, 2010.

[31] A. Corradi, M. Fanelli and L. Foschini, "VM consolidation: A real case based on OpenStack Cloud," Future Generation Computer Systems, vol. 32, pp. 118-127, 2014.

[32] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen and S. Shenker, Extending Networking into the Virtualization Layer., Hotnets, 2009.

[33] R. Fielding, "Representational state transfer," Architectural Styles and the Design of Netowork-based Software Architecture, pp. 76-85, 2000.

[34] Alliance, OSGi, Osgi service platform, release 3, IOS Press, Inc., 2003.

[35] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," Proceedings of the sixth conference on Computer systems, pp. 301-314, 2011.

[36] Felix, Apache, "Apache Felix-welcome," Apache Software Fundation, 12 03 2015. [Online]. Available: http://felix.apache.org. [Accessed 26 03 2015].

[37] P. Lessard, Linux process containment: A practical look at chroot and user mode Linux, 2003.

[38] Yaser Jararweh, T. Loai, A. Fadi and D. Fahd, "Resource efficient mobile computing using cloudlet infrastructure," IEEE Ninth International Conference on Mobile Ad-hoc and Sensor Networks (MSN), pp. 373-377, 2013 .

[39] Ha, K., Abe, Y., Chen, Z., Hu, W., Amos, B., Pillai, P., Satyanarayanan, M., Adaptive VM Handoff Across Cloudlets, Technical Report CMU-CS-15-113, Computer Science Department, Carnegie Mellon University, June 2015

Biographie :

☛ Charif Mahmoudi est a obtenu son Master et puis son Doctorat à l'université de Paris-Est. Il est actuellement chercheur à l'institut national des standards et technologies (NIST) qui relève du département américain du commerce. Il s'intéresse aux méthodes de spécification formelles des systèmes mobiles ainsi que les communications et réseaux dans le cadre du Cloud mobile ;

☛ Fabrice Mourlin a obtenu son doctorat en informatique en 1992 à l'université Paris Sud Orsay, ensuite il y a été maître de conférences pendant 10 ans avant de poursuivre à l'université Paris-Est où il a obtenu son habilitation à diriger des rechercher en 2008. Il est invité régulièrement au NIST. Son domaine de recherche porte sur la mobilité de code dans les systèmes distribués hétérogènes ;

☛ Guy Lahlou Djiken est doctorant / PhD Student en informatique à l'université de Paris-Est en cotutelle avec l'université de Yaoundé I, dans le cadre de MSTIC au sein du laboratoire LACL (Laboratoire d'Algorithme, de Complexité et de Logique). Il est membre de l'équipe de recherche Spécifications et Vérification de Systèmes et ses travaux de thèse portent sur la mobilité de code dans les systèmes embarqués.