Self-management of machine-to-machine communications: a multi-models approach

Cédric Eichler, Ghada Gharbi, Thierry Monteil, Patricia Stolf, Nawal Guermouche

Self-management of machine-to-machine communications: a multi-models approach

Cédric Eichler*, Ghada Gharbi, Thierry Monteil, Patricia Stolf and Nawal Guermouche

CNRS, LAAS,
7 avenue du Colonel Roche,
F-31400 Toulouse, France
and
IRIT,
118 Route de Narbonne,
F-31062 Toulouse, France
and
Univ de Toulouse,
UPS, F-31400, INSA, F-31400, UTM,
F-31100 Toulouse, France
E-mail: ceichler@laas.fr
E-mail: ggharbi@laas.fr
E-mail: monteil@laas.fr
E-mail: pstolf@laas.fr
E-mail: nguermou@laas.fr
*Corresponding author

Abstract: Machine-to-Machine (M2M) paradigm apply to systems composed by numerous devices sharing information and making cooperative decisions with little or no human intervention. The M2M standard defined by the European Telecommunications Standards Institute (ETSI) is the only one providing an end-to-end view of the global M2M architecture. Noticeably, it furnishes a standardised framework for inter-operable M2M services that satisfies most of M2M modelling requirements. However, and even though M2M systems usually operate in highly evolving contexts, this standard does not address the issue of system adaptations. It is furthermore unsuitable for building self-managed systems. This paper introduces a multi-model approach for modelling manageable M2M systems. Said approach consists in a formal graph-based model on top of the ETSI M2M standard, alongside bi-directional updates that ensure layer coherency. Its fitness for enforcing self-management properties is demonstrated by designing high-level reconfiguration rules. Finally, its applicability is illustrated and evaluated using a smart-metering application.

Keywords: M2M communicating systems; dynamic reconfigurations; autonomic computing; graph rewriting; ETSI M2M architecture.

Reference to this paper should be made as follows: Eichler, C., Gharbi, G., Monteil, T., Stolf, P. and Guermouche, N. (xxxx) 'Self-management of machine-to-machine communications: a multi-models approach', *Int. J. Autonomous and Adaptive Communications Systems*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Cédric Eichle is a PhD applicant at LAAS-CNRS and IRIT Laboratories (Toulouse, France). After obtaining a Certified Engineer degree in Computer Science and Applied Mathematics, he has been working on formal methods for the specification of dynamic distributed systems. He is currently co-managing the national ANR SOP project, for which he is also developing an autonomous manager based on correct by construction reconfigurations.

Ghada Gharbi is currently a PhD student at LAAS-CNRS Laboratory, Toulouse, France. She has been awarded a Master Diploma in Software Engineering and Multimedia Systems specialised in protocols and networks systems and received an Engineering Diploma in Computer Science from the National School of Computer Science of Tunisia (NSCS). Her research interests include the architectural reconfiguration of distributed communicating systems using formal models such as graph grammars and ontology involving concepts such as deployment, self-adaptability and communication models.

Thierry Monteil is an Associate Professor in Computer Science at INSA of Toulouse and Researcher at LAAS-CNRS. He has a Doctorate in parallel computing and Certified Engineer degree in Computer Science and Applied Mathematics. He works on parallel computing (LANDA parallel environment), grid resources management (AROMA project), internet of things (OM2M project), computer and network modelling, autonomous policies to improve performance on distributed applications and parallelisation of large electromagnetic simulation.

Patricia Stolf is an Associate Professor at the Toulouse University, France. She obtained a PhD in 2004 at LAAS-CNRS (Toulouse-France) on Tasks scheduling on clusters for remote services with quality of service. She is currently working in the IRIT Laboratory on the field of distributed algorithms and autonomic computing in large scale distributed systems. She studies resources management in energy and thermal-aware task scheduling and autonomic systems. She is involved in different research projects: the ACTION COST IC0804 'Energy efficiency in large scale distributed systems', the European CoolEmAll project and the national ANR SOP project.

Nawal Guermouche received her MS and PhD in Computer Science, respectively, in 2006 and 2010 from LORIA-INRIA, UHP University, Nancy 1, France. She is an Assistant Professor at INSA of Toulouse and a member of the SARA team at the LAAS-CNRS research laboratory. Her research interests focus on composition, analysis, reconfiguration and validation of service and component based systems. She has been involved in several transfer projects with industries (in France and Europe) for a wide range of applications including M2M.

This paper is a revised and expanded version of a paper entitled 'Graph-based formalism for machine-to-machine self-managed communications' presented at the 22nd IEEE International Conference on Collaboration Technologies and Infrastructure (WETICE), Hammamet, Tunisia, 17–20 June 2013.

1 Introduction

During the last years, the exponential expansion of wireless communications devices and the ubiquity of wireless communications networks have convey to the emanation of wireless machine-to-machine (M2M) communications as the most promising solutions to revolutionise the future 'intelligent' pervasive communications (Pandey et al., 2011).

Intrinsically, M2M systems are evolution prone as applications are stopped and started, machines discovered and shut down, etc. As most of its peers, the ETSI standard focuses on protocols and communications. It does not address the issue of dynamic reconfiguration or provide a suitable model for the reasoning required to build self-managed M2M architectures. These considerations belong to the field of dynamic software architectures that enables adaptation in autonomic distributed systems, coping with new requirements, new environments, and failures. To conciliate functionalities and manageability, we elaborate in this paper a component-based, bi-layered framework for modelling M2M systems. A graph-based representation built on top of the ETSI M2M standard constitute respectively the formal and functional layers of the framework. In order to ensure inter-layers coherency, the model also comprises bi-directional communications between these two layers. In this way, the model benefits from the best of both worlds. The graph-based characterisation allows the definition of consistency preserving reconfiguration mechanisms. On the other hand, it still possess the functionalities granted by the standard, such as discovery protocols and machine interoperability. We exploit this model by defining high level policies of reconfiguration, relying on graph rewriting, to enforce self-management properties.

The remainder of the paper is organised as follows. Section 2 investigates existing approaches for representing M2M systems in particular and approaches appropriate for the management of dynamic system in general. Since no single representation handle every modelling requirements of M2M systems, Section 3 proposes a multi-models, bi-layered, approach to meet said needs. The fitness of the resulting framework is demonstrated in Section 4, where the enforcement of high-level adaptation policies is discussed. Application of the framework to model and reconfigure an use-case is developed in Section 5. Finally, Section 6 is dedicated to conclusion and outlooks.

2 Related work

2.1 Machine-to machine systems and communications

The M2M paradigm is a broad label. It can be used to describe any technology that enables automated, wired or electronic devices. The aim of M2M systems is to allow devices to be interconnected, networked, and controlled remotely with low

cost, scalable, and reliable technologies. M2M systems and communications have been successfully exploited to enhance efficiency and to reduce (or even suppress) human intervention in a vast range of contexts. They intervene for example in the specification and the implementation of automotive, smart metering, eHealth, and smart grid applications.

On one hand, M2M systems possess common characteristics and requirements (Pandey et al., 2011), such as network heterogeneousness and dynamism, mobility and time sensibility of data, and device intelligence. On the other hand, M2M technical solutions are multiple, highly fragmented, and usually dedicated to single applications. Consequently, the M2M market development is critically slowed down, while the costs of development, maintenance, and research in M2M systems are increasing. To meet these challenges, standardisation is a key enabler to remove the technical barriers and ensures inter-operable M2M services and networks.

A significant number of standardisation bodies (TIA, http://tiaonline.org/standards; ESNA, http://www.esna.org/; IPSO, http://www.ipso-alliance.org/) are currently working on defining architecture and services standards to support M2M communication requirements. Some alliances focus on specific M2M areas and do not intend to cover every M2M characteristics. Among them, two approaches can be distinguished: from scratch (e.g., the OMA for device manageability) and improvement of existing standards to fit M2M requirements [e.g., promoting internet protocol (IP) for smart objects]. Furthermore most works are still in preliminary stage and provides only drafts (e.g., the TIA working group, IETF).

The European Telecommunications Standards Institute (ETSI, http://www.etsi.org/Website/Technologies/M2M.aspx) M2M group is the only one that has developed an end-to-end view for M2M communications. Its standards facilitate the deployment of vertical applications and the innovation across industries by exposing data and providing services. This architecture is defined in several technical reports fixing (ETSI TR, http://www.etsi.org/technologies-clusters/technologies/m2m; ETSI FA TR, 2011):

1 functional and behavioural requirements of each network element to provide an end-to-end view

2 the functional architecture with the different M2M services capabilities

3 the protocols of various interfaces.

However, the ETSI M2M standard does not address the issue of system adaptations, even though M2M systems usually operate in highly evolving contexts. Besides, the very model constituted by these specifications appears unfit for the specification of self-management operations. Multi-models approaches (Roh et al., 2004; Sharrock et al., 2008; Loulou et al., 2004) can successfully be applied when no single representation meet all the requirements for modelling a system. To enable the management of M2M systems, concepts of the ETSI M2M standard can be mapped unto an appropriate model, the result benefiting from both representations.

2.2 Models for dynamic systems management

Concerns, models, and approaches discussed here are not necessarily related to M2M systems. Rather, their merits regarding management are investigated to identify the one

that should be integrated in the multi-models approach required to cover both functional and management-related aspects of M2M systems.

Dynamic software architectures are studied for handling adaptation in autonomic distributed systems, coping with new requirements, new environments, and failures. Particularly, the description of evolving architectures cannot be limited to the specification of a unique static topology. It must characterise the scope of all acceptable configurations. This scope characterises an architectural style, qualifying what is correct and what is not. Naturally, once this distinction made, the question of specification of the modifications themselves arises.

Model-based approaches furnish very intuitive and visual formal or semi-formal description of structural properties (Bradbury et al., 2004), and system evolution. Designing and describing software models using UML, for example, OMG, UML (2005) is a common practice in the software industry. It provides a standardised definition of system structure and terminology, facilitating the understanding of the architecture (Selonen and Xu, 2003). Nevertheless the generic fitness of model-based approaches may imply poor means of describing specific issues like behavioural properties. Therefore, they are often coupled with description using architecture description languages (Roh et al., 2004; Sharrock et al., 2008), mapping the concepts of architecture description languages into the visual notation of UML, or other formalism (Loulou et al., 2004). In spite of its wide acceptance, UML-based descriptions appear to lack expressiveness and formal tools for guaranteeing consistency, due to the inherent semi-formalness of UML. Formal unambiguous methods are necessary to study the consistency of a system at a given time (i.e., its compliance to an architectural style). To efficiently tackle correctness in the scope of dynamic reconfiguration, correctness by construction through formal approaches have emerged (Bonakdapour et al., 2010). *Based on formal proofs and reasoning in design-time, they guarantee the correctness of a system, requiring little or no verifications in run-time*. A way to achieve such proofs is to investigate the properties of transformations with regard to consistency preservation, so as to ensure that if a transformation is applicable on a correct configuration its result is another correct configuration.

Graph-based methods for software modelling are appropriate for conceiving correct by construction frameworks. Theoretical work in this field provides formal means to specify and check structural constraints and properties (Rozenberg, 1997; Bruni et al., 2008). Within this kind of approaches, some methods are restricted to the usage of type graphs alone (Wermelinger and Fiadeiro, 2002) and suffer from the same lack of expressiveness as UML-based methods. Other works (Hirsch et al., 1999; Le Metayer, 1998) are based on graph grammar, or graph rewriting system, offering a generative definition of the scope of correctness. In this context, graph rewriting rewriting rules intervene in the very definition of the style. This tight link with the scope of correctness makes these rules fit for the specification of consistency preserving reconfigurations. These advantages motivate the adoption of graph-grammar-based descriptions as a management-centric model, lifting the identified restriction inherent to the consideration of a functional representation alone.

3 A multi-models, bi-layered, approach

This section describes the proposed approach, its layers, and inter-layers communications.

In the ETSI vision, M2M constituents are seen as resources. They are defined in a tree structure and handled with the RESTful style of data exchange. This resource tree consists on a logical grouping of resources that allows simple addressing of resources, flexibility in exchanging application data and simple APIs. The formal layer is composed by a generic graph grammar to enable the definition of consistency preserving reconfigurations.

These two models characterise any M2M architecture. The management of actual M2M architectures shall depend on their instantiations. Furthermore, only a subset of functional properties is required to enforce the aimed management mechanisms. Consequently, it is not relevant to duplicate every single piece of information contained in the functional layer onto the formal one. Similarly, information required to conduct adaptation may not be needed to guarantee the functioning of the system (e.g., transmission delay).

Considering two layers impose guaranteeing theirs coherence. Since the system is dynamic, layers will have to evolve to fit its current state. Besides, evolutions can originate from both layers, since discovery and reconfigurations are conducted respectively in the functional and formal one. To ensure the coherence of the two layers, changes must thus be bi-directionally impacted. Bi-directional updates and communications described in this section ensure this crucial property.

3.1 Functional layer based on the ETSI M2M standard

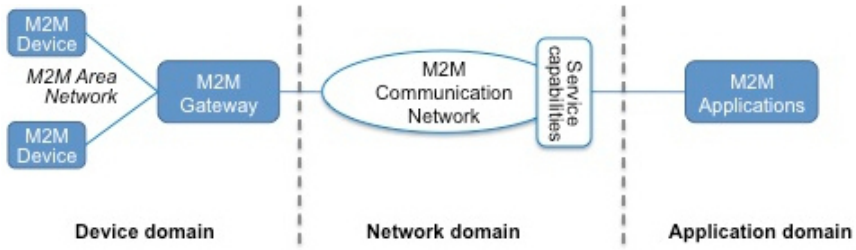
The ETSI has divided M2M systems, as shown in Figure 1, into three domains:

- The *application domain* runs the service logic and uses M2M services capabilities accessible via an open interface. The application data is referred as resources. Resources are defined in a tree structure and handled with the RESTful style of data exchange.
- The *M2M device domain* includes data end points such as sensors, smart meters, microprocessors, and gateway.
- The *network domain* is a network technology providing connectivity between M2M devices from the M2M device domain.

As stated previously, M2M constituents are represented by a resource tree structure. This structure exposes functions through a set of open interfaces grouped under the heading ‘M2M service capabilities layer (SCLL)’. This last comports:

- *NSCL*: network service capabilities layer refers to M2M service capabilities in the network domain
- *GSCL*: gateway service capabilities layer refers to M2M service capabilities in the M2M gateway
- *DSCL*: device service capabilities layer refers to M2M service capabilities in the M2M device

Figure 1 ETSI-Simple-M2M-Architecture (see online version for colours)



- *SCL*: service capabilities layer, refers to any of the following: NSCL, GSCL, DSCL.

In M2M systems, data come from a large number of devices and are exchanged between various entities (i.e., applications) through data containers. These lasts are used as mediators taking care of buffering the data. They make the exchange abstracted from the need to set direct connections and allow for scenarios where both parties in the exchange are not online at the same time. To enable interactions between the distributed applications and devices, the registration and the announcement of resources must be fulfilled.

Registration comprises two sub-cases: *SCLs registration* and *Applications registration*.

- *SCLs registration* defines the procedures that allow a GSCL or a DSCL to register to the NSCL once discovery has been performed. SCL registration is a necessary procedure allowing the SCL to start resource management procedures. For scalability sake, peer-to-peer SCLs registrations (gateway/gateway, device/device, device/gateway) are also considered in this paper.
- *Applications registration* defines the set of procedures that allows an application to register to its local SCL. It is a necessary step for an application to be known and to start exchanging data using the Restful procedures.

Announcement Defines the set of procedures for a resource to advertise for a remote SCL. In order to enable the announcement of a container or an application on a remote machine, it is necessary that the host and target machines are mutually registered.

3.2 Formal layer based on graph rewriting and graph grammars

Before characterising M2M systems using a graph grammar, general concepts related to graph rewriting are introduced.

3.2.1 Graph rewriting rule and graph grammars

A configuration of a system captures its state at a given time. It can be modelled using attributed graphs whose vertexes specify entities (e.g., devices, applications, containers) and edges represent theirs relationships (e.g., registration deployment, utilisation).

Definition 1: Attributed graph

An attributed graph G is defined by the tuple (V, E, ATT) where:

- V and E $\subseteq$ V^2 correspond respectively to the set of vertexes and edges of G.
- ATT is a family of set indexed by V $\cup$ E. A set of this family is a sequence of couple (A, D_A) where A and D_A are respectively the attribute value and domain of definition. Accordingly, A is either a constant in D_A, noted between quotations marks, or a variable that may take any value in D_A.

An architectural style can be formalised using a graph rewriting system or graph grammar. Such systems are based on graph rewriting rules that require to identify common sub-structures by the mean of morphisms. Particularly here, induced sub-graph isomorphisms between graphs are considered. It should be stressed out that this relation is not symmetric and that the 'induced sub-graph' part of the denomination refers only to the second graph. In other words, the existence of an induced sub-graph isomorphism between two graphs G and G' means that there is an induced sub-graph of G' isomorph to G. An unattributed induced sub-graph isomorphism i between two graphs G = (V, E, ATT) and $\overline{G}$ = ($\overline{V}$, $\overline{E}$, $\overline{ATT}$) is defined as a mapping of V into a sub-set of $\overline{V}$ so that if there is an edge between two vertexes of G, there is an edge between their images in $\overline{G}$ and reciprocally (Rozenberg, 1997).

Definition 2: Unattributed induced sub-graph isomorphism

An unattributed induced sub-graph isomorphism i between two graphs G = (V, E, ATT) and $\overline{G}$ = ($\overline{V}$, $\overline{E}$, $\overline{ATT}$) is defined as an injective function f: $V \rightarrow \overline{V}$ so that $\forall$ (v, $\tilde{v}$) $\in E^2$, (v, $\tilde{v}$) $\in E \Leftrightarrow$ (f(v), f($\tilde{v}$)) $\in \bar{E}$.

By abuse of notation, for any vertex of G, i(G) refers to f(G). To tackle attributes, we impose firstly that two vertexes or two edges associated through an isomorphism have the same number of attributes. Attributes of two associated elements are themselves correlated with regard to the order of their occurrences. Identified attributes should have the same domain of definition. Secondly, identifications of attributes should be consistent (e.g., a variable should not be identified with two different constants). Therefore, a system of equations is built and the existence of an attributed induced sub-graph isomorphism is conditioned by its resolvability.

Definition 3: Attributed induced sub-graph isomorphism

There is an induced sub-graph isomorphism iso between two attributed graphs G = (V, E, ATT) and G' = (V', E', ATT'), noted G $\xrightarrow{iso}$ G' or simply G $\rightarrow$ G', if and only if there is an unattributed induced sub-graph isomorphism iso' from (V, E) to (V', E') such as

1 $\forall$ v $\in$ V (resp. $\forall$ e = ($\bar{v}$, $\tilde{v}$) $\in$ E^2), $|ATT_v| = |ATT_{iso'(v)}|$
(resp. $|ATT_e| = |ATT_{(iso'(\bar{v}),iso'(\tilde{v})}|$)

2 $\forall$ v $\in$ V (resp. $\forall$ e = ($\bar{v}$, $\tilde{v}$) $\in$ E^2), $\forall$ i $\in$ [|1, $|ATT_v|$|], $D_v^i = D_{iso'(v)}^i$
(resp. $D_e^i = D_{(iso'(\bar{v},iso'(\tilde{v})}^i$)

3 the system of equations S = { A = A' | ($\exists$ v $\in$ V, $\exists$ i $\in$ [|1, |ATT$_v$|], A = A$_v^i$ $\wedge$ A' = A$_{h(v)}^i$) $\vee$ ($\exists$ e = ($\bar{v}$, $\tilde{v}$) $\in$ E, $\exists$ i $\in$ [|1, |ATT$_e$|], A = A$_e^i$ $\wedge$ A' = A$_{(h(\bar{v},h(\tilde{v}))}^i$) } has at least one solution.

where A and A' are mute variable names, see Definition 1.

Solving the system of equations S results in identifying the value of some attributes with some constants in their domains of definition and/or with the value of some other attributes. Integrating the affectation obtained by solving the systems refers to the update of the value of the attribute to reflect these identifications. For example, if ((x,y), (x, "2") $\in S^2$, meaning that x has been identified to the variable y and the constant "2", integrating the affectations obtained by solving S will lead to replacing each occurrence of x and y by "2". For genericness sake, we define the following super-patterns.

Definition 4: Super-pattern

A super pattern is one of the following elements:

- A vertex whose only attribute is 'any', its domain of definition begin of no interest. Its attributes do not take part in the conditions 1, 2 or 3. It is only relevant in the phase where an unattributed sub-graph isomorphism is looked for.
- An attribute taking value in a subset of its domain of definition, materialised by enumerating the possibility, e.g., ("a" or "b", {"a", "b", "c"}). Such an attribute impacts the condition 3 by adding a constraint on the system of equation S.

The characterisation of graph rewriting rules used in this paper is based on the Double PushOut (Ehrig, 1987) approach.

Definition 5: Graph rewriting rule

A graph rewriting rule is a triplet (L, K, R) where L and R are two graphs, and K-called the Inv zone- is a sub-graph of both L and R. (L\K) is called the Del zone and (R\K) is called the Add zone. A rule is applicable on a graph G if there is an induced sub-graph isomorphism i: L $\rightarrow$ G and its application does not lead to the apparition of any dangling edge. Its application consists in erasing (L\K) and adding an isomorph copy of (R\K) integrating the affectation obtained by solving the system of equations related to i.

In this paper, graph rewriting rules are illustrated using the delta representation, where only one graph is considered. This graph is visually partitioned into three zones, from left to right the Del, Inv and Add zones.

Figure 2 An example of graph transformation

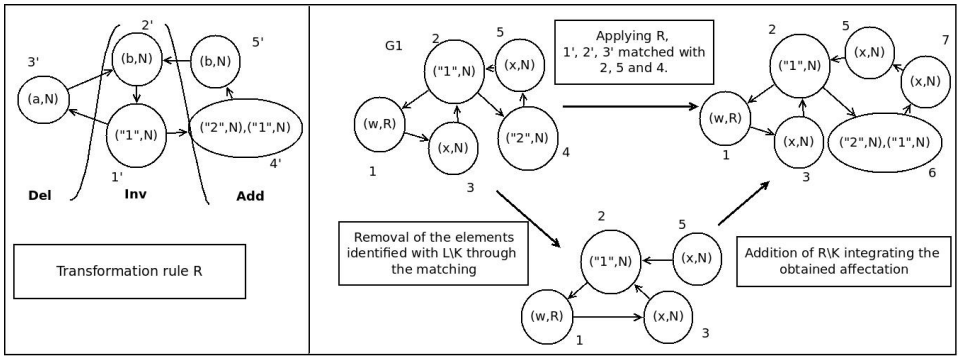


Figure 2 offers an example of how a transformation is handled in the previously defined approach. To lighten the figure, the attributes of the edges have not been represented and will all be considered equals. There exists an induced sub-graph isomorphism iso associating 1", 2" and 3" with respectively 2, 5 and 4. It identifies b with x and a with "2", these identification introducing no inconsistency. This morphism is such as L $\xrightarrow{iso}$ G1 and $\forall$ v $\in$ $V_{G1} \setminus$ iso(V_K), $\forall$ v' $\in$ $V_L \setminus V_K$, (v, iso(v')) $\notin$ E_{G1} $\wedge$ (iso(v'), v) $\notin$ E_{G1}, the deletion of the graph identified with Del through iso would not lead to the apparition of a dangling edge. The transformation R can be applied to G1 with the matching iso. The graph corresponding to the Del zone is removed and an isomorph copy of the Add zone is then added , with b being replaced by x to impact the identifications.

Inspired from Chomsky's (1956) generative grammars, graph grammars specifying an architectural style are defined as follows.

Definition 6: Graph grammar and their instances

A graph grammar is defined as a system $< AX; NT; T; P >$, where AX is the axiom, NT is the set of the non-terminal vertexes, T is the set of terminal vertexes, and P is the set of graph rewriting rules, also called grammar productions. An instance belonging to the graph grammar is a graph G such as there is not any nt $\in NT$ such as nt $\rightarrow$ G and is obtained starting

Thanks to the very definition of compliance to an architectural style characterised by a graph grammar, and in particular its generative aspect, consistency preserving reconfigurations can be built from the productions rules. Correct by construction reconfigurations is a key advantage of graph grammars.

Consider any rewriting rule r whose application is equivalent to the application of a production or a sequence of productions of the grammar, noted p in this paragraph. Note that we can consider a single production even in the case of a sequence, through composition. Trivially, r preserves consistency if its applicability conditions are equivalent or stronger than p's ones, e.g. if r requires a larger pattern to be found meaning that L_r is a sub-graph of L_p.

At first sight, we should be able to terminate anything that has been started. Such rules can be obtained from the productions using graph rewriting rules' property of reversibility. Let's consider a consistent instance of a graph grammar, constructed by

applying the sequence of productions rules $(p_i)_{i\in[|1,N|]}$ to the axiom. Intuitively, if a rule r is applicable, the relationship or entity it terminates has previously been started, meaning that there exists k $\in [|1, N|]$ such as r is the reverse of p_k. r preserves consistency if r and each rule in $[|k, N|]$ are sequence independent.

3.2.2 Formal characterisation of M2M systems

The formal layer, introduced to reason and manage actual M2M applications, is composed by a generic graph grammar. This last characterises M2M systems. Application built according to the M2M paradigm are instances of the ETSI M2M standard. In a similar fashion, management of actual M2M architectures shall rely on graph grammars instantiated from the meta-graph grammar.

For conciseness sake, the pieces of information considered here are restricted to:

- The deployed devices and the kind of applications they may run. We also keep track of which devices 'see each other' (i.e., are registered to one another) and the propagation delay due to the physical network through which they communicate.
- The deployed containers, on which device, and the devices they are announced to.
- The deployed applications, on which device, their type, the devices they are announced to, the containers they currently use, and how they use each containers.

When building a graph grammar, the first thing to consider is the set of terminal terms. These lasts characterise the kinds of entities constituting an M2M system, and the information that has to be carried by each entity. Each vertex is identified by a unique identifier, the set of identifier being noted Id. By convention and to simplify the notations, a vertex v attributed by ATT_v will be noted v(ATT_v). In M2M systems, and considering the previous paragraph, we distinguish three arch-entities representing the vertexes modelling:

- The deployed containers, simply noted $V_{containers}$((id,Id)), and refereed to as $V_{containers}$.
- The deployed applications and their type. Let applicationType be the set of application's type. The considered arch-vertex is $V_{applications}$((id,Id), (appliType, applicationType)).
- The network, gateways and ETSI devices that do not require a gateway to communicate, and the type of applications that may run. Let applicationTypes be the power set of applicationType. This kind of vertexes is represented by the arch-vertex $V_{devices}$((id,Id),(deviceType, {"Network", "Gateway", "ETSIdevice"}), (runnableApplication, applicationTypes)).

Once the entities defined, one should consider their relationships, modelled by edges' attributes:

- "registered": two instances of the device arch-vertex may be registered to one another
- "created": between a container or an application and the device it is deployed on

- "announced": between a container or an application and a remote device.

The conditions that have to be met for a relationship to be initiated, or for an entity to be deployed, as well as how to do it, are described by the productions of the grammar. For readability sake and considering that there is no ambiguity on domains of definition, they are implicit in the following. Furthermore, only the most representative productions are graphically represented.

The production p_1, illustrated in Figure 3(a), describes the initialisation of an M2M network consisting in the deployment of the vertex representing the network. The addition of a device, be it a gateway or an ETSI device, is managed by the rule p_2, represented in Figure 3(b). Such a device has to be registered to the network and see the resources of the network in a similar fashion.

The rule p_3, illustrated in Figure 3(c), formalises the addition of an application. The rule p_4 modelling the deployment of a container is similar and thus no represented. Note that the device should be able to run such an application (i.e., appli $\in$ runnableApplis).

The rule p_5, presented in Figure 3(d), depicts the peer to peer registration of an ETSI device or a gateway to another one. Registration is a symmetric relation. Attributes $type1$ and $type2$ are variable and can take any value in {"Network", "ETSIdevice", "Gateway"}. $type1$ and $type2$ can be replaced by "ETSIdevice" or "Gateway", since neither type1 nor type2 should be equals to "Network". However, each device and each gateway registers to the network when deployed, and remains registered. Since there is no edge between the two vertexes in the invariant part of the rule, it is not possible to find an induced sub-graph isomorphism with $type1$ or $type2$ equals to "Network".

The announcement to a device 2, of an application or a container deployed on a device 1, requires devices 1 and 2 to be registered as shown by the production p_6 depicted in Figure 3(e).

An application may use a container, i.e., reads and/or writes on it, if one of the following conditions is met:

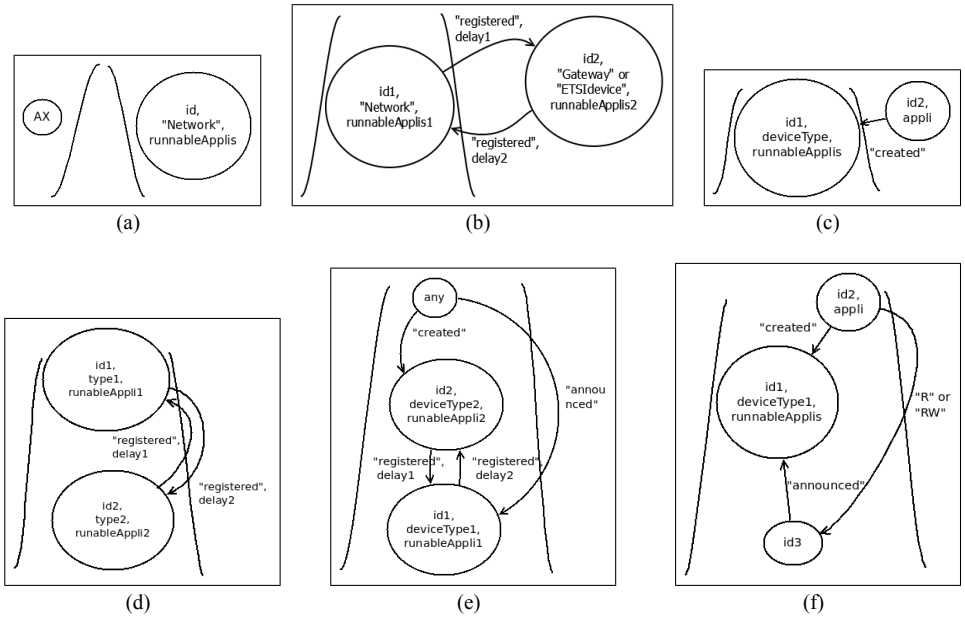
- Both are deployed on the same device. This very simple case is described by the production p_7, not represented here.
- The application is running on an entity on which the container is announced, p_8, illustrated in Figure 3(e). Note that, since a container can not be announced on the device on which it is deployed, the application uses a remote container.

Finally, the graph grammar characterising M2M systems is
$< AX, \emptyset, \{V_{containers}, V_{applications}, V_{devices}\}, \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\} >$.

3.3 Inter-layers communications and updates

Both layer can originate a model evolution, since discovery is conducted in the functional layer and reconfigurations in the formal one. To ensure the coherence of the two layers, changes must thus be bi-directionally impacted.

Figure 3 Most representative productions of the grammar, (a) initialisation (p_1) (b) addition of a device (p_2) (c) addition of an application (p_3) (d) registration of a device (p_5) (e) announcement of an application or a container (p_6) (f) an application uses a distant container (p_8)



3.3.1 From the functional layer to the formal one

Whenever a new entity joins the system or is started, it is spotted by the functional layer through its discovery protocol. Three types of entities are considered: containers, applications and devices. A container or an application can be discovered simultaneously with the device they are deployed on, or on an already know device. In any case, discovered devices are firstly treated, applications and containers being treated thereafter. No specific order of treatment needs to be adopted within these two sets.

- *Device discovery:* A message is sent with the unique identifier of the device, the types of application it may run, and its nature (i.e., gateway or ETSIdevice). The graph is updated by applying the rule p_2, $id2$, "Gateway" or "ETSIdevice" and $runnableApplis2$ instantiated with the previous information.
- *Container discovery:* A message is sent with the unique identifiers of the container and the device it is deployed on. The rule p_4 is then applied to the formal layer, with $id1$ and $id2$ fixed to the received identifiers.
- *Application discovery:* A message is sent with the unique identifiers of the application and the device it is deployed on, as well as the type of the application. The graph is updated by applying the rule p_3 with $id1$, $id2$ and $appli$ fixed to the received values.

3.3.2 From the formal layer to the functional one

Whenever an action is applied consequently to a decision in the formal layer, the implication must be impacted on the real system (i.e., the functional layer). Since reconfigurations are modelled by graph rewriting rules, they can be decomposed regarding each vertex and edge in the Add and Del zones. An action will be performed for each element depending on its zone and its attributes. These operations are conducted in the following order for addition: devices, containers, applications, registration, announcement and utilisation. If linked to deletion, they are operated in the opposite order. Neither deletion nor addition have the priority with regard to the other, and there is no order within a type (e.g., if several applications are simultaneously started, they may be treated in any order).

- *Device*: Suppression and addition of devices on the formal layer does not leads to the suppression and addition of devices in the functional layer. Rather, it will lead to a change in the status of the device notifying whether it is active or not. This mechanism ensures that registrations of the devices, and remote information on the device are not lost when it is stopped.

    1 *activation*: an UPDATE call with the status ‘active’ is sent in the functional layer to the resources tree of the concerned device

    2 *deactivation*: an UPDATE call with the status ‘idle’ is sent in the functional layer to the resources tree of the concerned device.

- *Application and/or container*:

    1 *addition*: a CREATE call with all required information is sent to the functional layer, more precisely to the resources tree of the device on which the application or the container is deployed

    2 *deletion*: a DELETE call with all required information is sent to the resources tree of the device the application or the container is deployed on.

- *Registration/de-registration*: In this part, peer-to-peer interaction capabilities are treated, which, as a unique feature, significantly improves system scalability and performances by enabling gateway/gateway, device/device, and device/gateway direct communications.

    1 *Registration*: To add a peer-to-peer communication, a CREATE<scl> call is sent from a device to another (Hosting device) in order to be able to interact with it directly.

    2 *De-registration*: To delete a peer-to-peer communication, a DELETE<scl> call shall be used by the issuer device to de-register from the remote device. When a device registers to another device, two resources are created, one in the issuer and another in the hosting. The de-registration process shall consist in the deletion of both resources previously created.

- *Announcement*:
  1. *new announcement*: a CREATE call of the resource 'applicationAnnc' or 'containerAnnc' with all required information is sent to the functional layer, more precisely to the resources tree of the device where the resource will be announced
  2. *de-announcement*: a DELETE call of the resource 'applicationAnnc' or 'containerAnnc' with all required information is sent to the resources tree of the device where the resource will be de-announced.
- *Utilisation*: If an application has to read and/or write on a container, or to stop doing so, an UPDATE call with the corresponding application requesting entity is sent to the container to update the access rights accordingly.

4 Enforcement of self-management policies

This section exploits the proposed framework and demonstrates its fitness by defining self-management policies for any M2M system. In the following, we assume the existence of a monitoring and/or an analysing routine able to throw the following events:

- a container c has been accessed more than x times by distant applications in an interval of time t
- there is less than x% of battery left on a device d.

Each event triggers an algorithm using graph rewriting rules as described below. In concordance with the remark on graph grammars and correctness of reconfigurations, the application of the rewriting rules used in reconfiguration scenarios is equivalent to the application of a production or a sequence of productions of the grammar, or the reverse of a production. This ensures that the system stays in a state constructible with a sequence of productions, and thus that the reconfiguration is correct.

The graph representing the formal layer when an event is thrown is noted G = (V, E, ATT). In this section, 'update the functional layer' refers to the processes described in Section 3.3.2.

When 'a container c has been accessed more than x times by distant applications in an interval of time t', it should be moved to the network in order not to saturate the communication channel of the device where c is deployed. Each application that reads and/or writes on c is redirected to the corresponding container. These actions are described in the algorithm *migrate(idC, idD)*, where idC and idD are respectively the identifiers of c and of the device where the new container shall be deployed, in this case the network.

migrate(idC, idD)
createNannouce(idC, idD)
for each induced sub-graph isomorphism i: $L_{redirect(idC,idNewC)} \xrightarrow{i} G$
apply graph rewriting rule redirect(idC, idNewC) w.r.t. i
update the functional layer
apply graph rewriting rule destroy(idC)
update the functional layer

With *createNannounce(idC, idD)* being the process first creating a new container on the device identified by idD. Then, each registration and announcement required prior to the redirection of applications using idC are conducted.

createNannounce(idC, idD)
apply graph rewriting rule p_4 with id1 fixed to idD
update the functional layer
idNewC $\leftarrow$ id2, the id of the new container
for each induced sub-graph isomorphism i: $L_{registerD(idC,idNewC)} \xrightarrow{i} G$
apply graph rewriting rule registerD(idC, idNewC) w.r.t. i
update the functional layer
for each induced sub-graph isomorphism i: $L_{announceC(idC,idNewC)} \xrightarrow{i} G$
apply graph rewriting rule announceC(idC, idNewC) w.r.t. i
update the functional layer

where *redirect(idC,idNewC)*, *destroy(idc),registerD(idC,idNewC)* and *announceC(idC, idNewC)* are described in Figure 4. Note that the uniqueness of the induced sub-graph isomorphism, with regard to which p_4, p_6, *duplicate(idC, idNewC)* and *destroy(idC)* are applied, is ensured by the uniqueness of the identifier of the container.

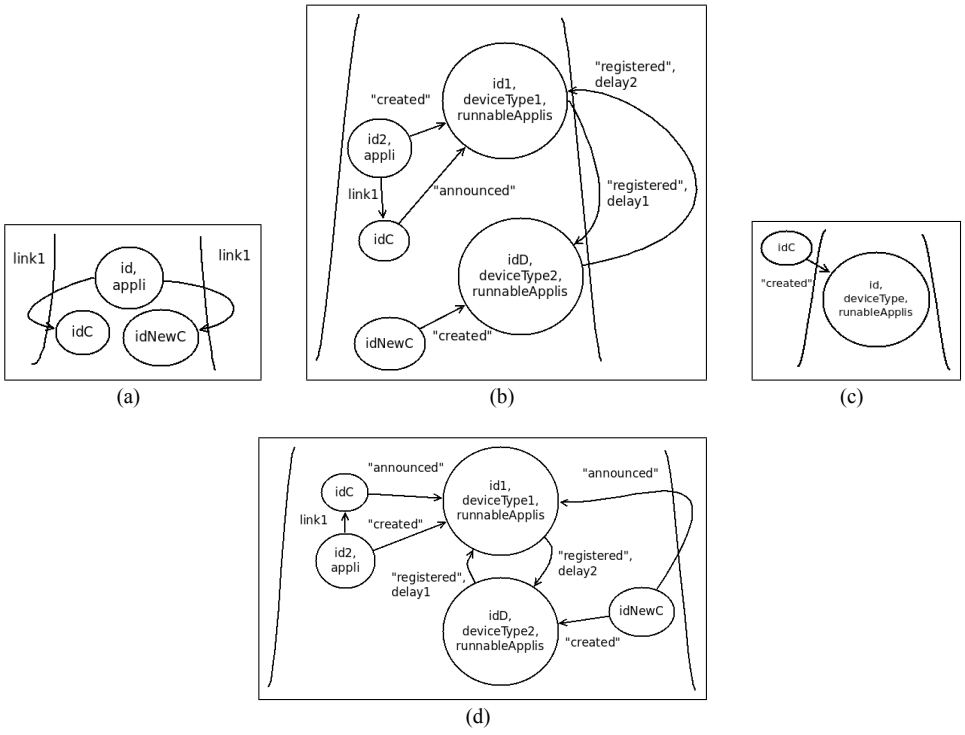
The case where 'there is less than x% of battery left on a device d', may lead to the loss of data in the containers deployed on the device d whenever it will shut down due to an empty battery. In order to prevent this loss, each container deployed on d is moved elsewhere. The targets of these migrations impact the configuration quality and should therefore be chosen carefully. To ensure service continuity, each application that reads and/ or writes on a migrated container is redirected to the corresponding container. These steps are conducted by the process *backup(idD)*.

backup(idD)
for each induced sub-graph isomorphism i: G'(idD) $\xrightarrow{i}$ G
idC $\leftarrow$ the identifier of the container associated with id through i.
idTargD $\leftarrow$ findSuitableDevice(idC)
migrate(idC, idTargD)

With G' being nothing more than a container deployed on device, or R_{p_4}.
The function *findSuitableDevice(idC)* returns the identifier of the device on which a container $\tilde{c}$ should be deployed in order to replace or reinforce the container c identified by idC.

Intrinsically, the faster the container usage, the better. Consequently, the most suitable location for a container is considered to be the one minimising the sum of the transmission delays from each user (i.e., applications using the container). In the present case, future users are applications using c, since they will eventually use $\tilde{c}$ instead. Furthermore, an application can use a container only if their respective locations are announced to each other. Consequently, there should be an edge between the each device where an application using c is deployed and the target of the migration.

Figure 4 Rules intervening in *migrate(idC, idD)*, (a) *redirect(idC, idNewC)*: redirection of an input and/or output of an application (b) *registerD(idC,idNewC)*: registration of a device on which an application to be redirected is deployed (c) [*destroy(idc)*: suppression of the original container (d) *announceC(idC, idNewC)*: announcement of the new container



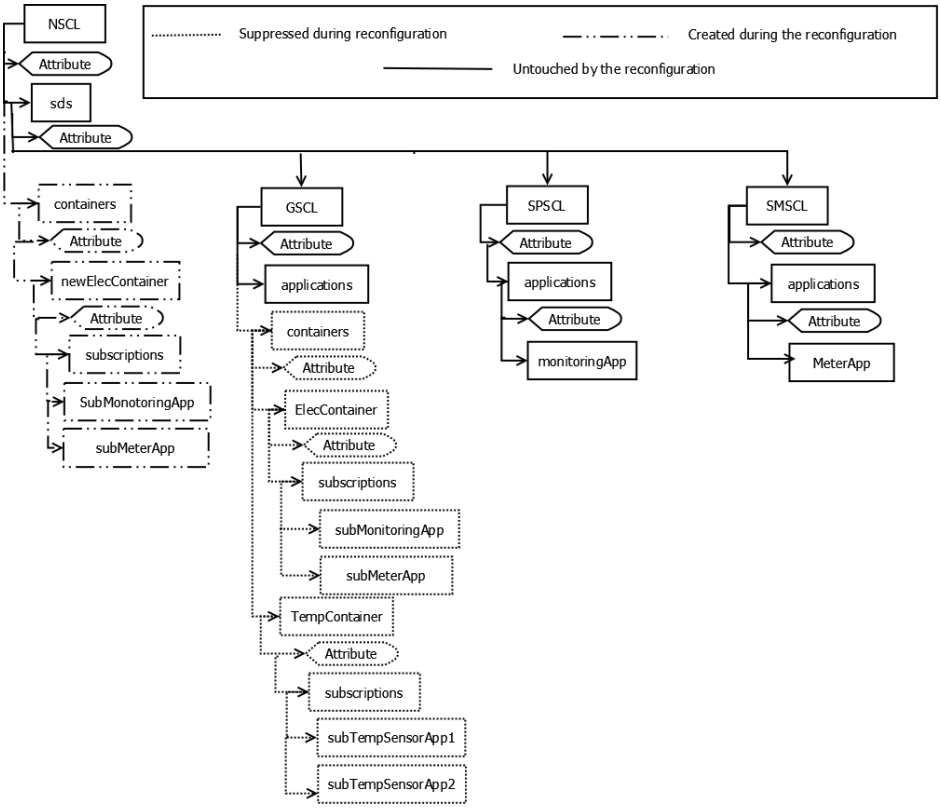
In real-life system, it is highly probable that the appropriate device is currently running an application using c, since they nullify a term of the sum to be minimised. Accordingly, the potential targeted devices are restricted to the ones on which an application using c is deployed (except the one where c is deployed) plus the network. Considering this last ensures that at least one location can be seen from any other. This set can be constructed by looking for induced sub-graph isomorphisms from R_{p8} to G. It is supposed to be known and noted *potential(idC)*, while *potential_id(idC)* qualifies its set of identifiers.

findSuitableDevice(idC)
searchGraph ← the sub-graph of G induced by potential(idC)
for each $i \in$ potential_id(idC)
if $\forall\ id \in potential\_id \backslash \{i\}$, ($i$,$id$) $\in \mathrm{E}_{searchGraph}$
sumFrom(i) ← $\sum_{id \in potential\_id \backslash \{i\}} (\mathrm{A}_{searchGraph})^2_{(i,id)}$
else sumFrom(i) ← ∞
$idD \leftarrow id$ such as sumFrom(id) = $\min_{i \in potential\_id}$ sumFrom(i)
if sumFrom(idD) $\neq \infty$ return idD
else return $idNetwork$

5 Smart metering use case

Throughout this section, smart metering systems are used to illustrate and evaluate the application of the described approach to a real-life use case. An advanced metering infrastructure (AMI) contains numerous heterogeneous machines (e.g., meters, sensors, actuators, gateways, processors, smart phones). They are interconnected and cooperate to achieve clearly defined objectives such as house and cities monitoring and billing.

Figure 5 Network resource tree

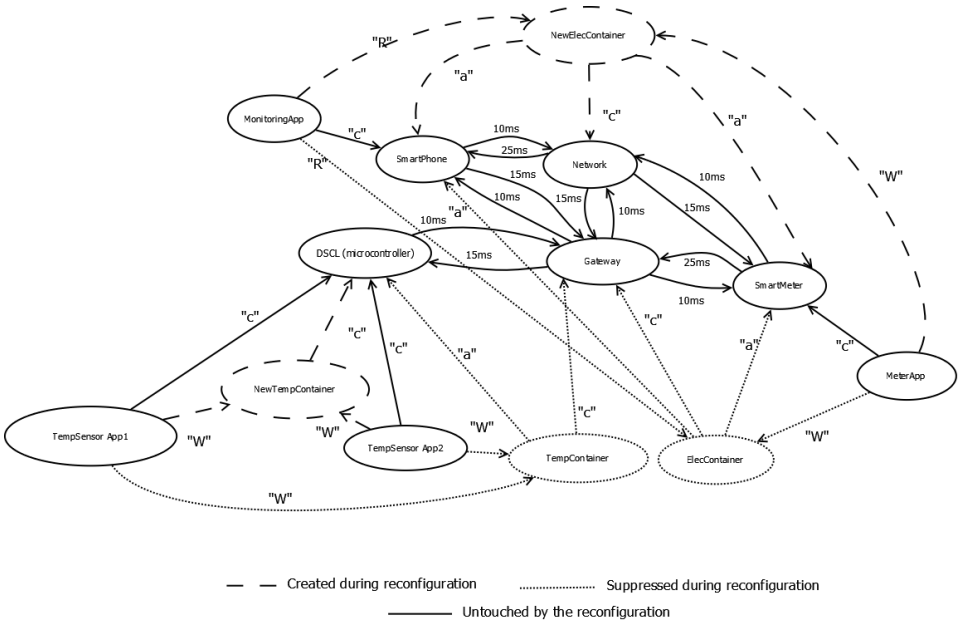


5.1 Example

In the present scenario, we consider the reconfiguration of an AMI monitoring a house. The initial and final graphs modelling the application is represented in Figure 6. The network resources tree, a representative part of the functional layer, is illustrated in Figure 5. To clarify the figures, vertex attributes are implicit while "created" and "announced" have been shortened to "c" and "a". The house is equipped with the following devices: a microcontroller connecting temperature sensors, a gateway, a smart

phone, and a smart meter recording the electric consumption of the house. These devices are connected to the network.

Figure 6 The formal layer: an instance of the graph grammar depicting the state of the system



Initially, the gateway is dedicated to 'electricity' and 'temperature' containers due to its high storage capacity. The temperature sensors and the metering application write on the 'temperature' container and the 'electricity' one, respectively. The monitoring application is deployed on the smart phone and reads data from the 'electricity' container.

To illustrate self-management policies, let's assume that the gateway has a low battery and will soon breakdown. In order to prevent potential data loss, its containers will be migrated using *backup(idGateway)*. The formal layer will receive an http request from the functional one (where the monitoring takes place) with the format *PUT(backup, idGateway)*. According to the policy, *migrate(findSuitableDevice)* will be applied to 'electricity' and 'temperature' containers. Firstly, the migration targets are chosen. This is conducted by finding the device visible from any device on which an application using the container is deployed and minimising the sum of transmission delays. The 'temperature' container is used by two applications executed on the microcontroller, namely temperature sensor 1 and 2. This container should thus be migrated to the microcontroller. This enables past temperature measurements not to be lost, and future to still be backed up. The 'electricity' container, however, is used by the meter and monitoring applications which are located on two distinct devices. The two devices are not registered to one another so that the container may not be deployed on either of them. Consequently, the Network is chosen as the migration target.

The container migrations take place using *migrate(idTemperatureContainer, idMicrocontroller)* and *migrate(idElecrticityContainer, idNetwork)*. Two containers, newTemperatureContainer and newElectricityContainer, copies of the previous ones are created on the chosen devices (i.e., the microcontroller and the network, or more precisely a server in the network). No registration is necessary, but newElectricityContainer is announced to the smart meter and the smartphone according to *announce*. Applications are then redirected, and old containers removed.

5.2 Evaluation results

The objective of this section is to prove the feasibility and study the scalability of the proposed approach. To implement the transformations rules intervening in the reconfigurations policies, we used the graph matching and transformation engine (GMTE, http://homepages.laas.fr/khalil/GMTE/).

The scenario described in the previous sub-section has been conducting with various number of applications and containers. We started by the configuration described above deploying two containers on the gateway and four applications. To evaluate scalability, the number of applications using containers has been increased from 4 to 50. We considered two, three, and four containers deployed on the gateway. To experiment the dynamic reconfiguration approach, we simulate the worst case of the gateway breakdown. Containers are migrated to suitable locations and corresponding applications are reconfigured to read/write from the new containers locations. The experimental performance evaluation focuses on the execution time. The results, shown in Table 1, demonstrate that the execution time to handle dynamic reconfiguration is in the order of seconds. These results remain acceptable in particular if we consider that

1 the case where 50 applications simultaneously use containers deployed on a unique gateway is very infrequent
2 the execution time comprises decision making optimising the resulting configuration, reconfiguration itself , and the guarantee that the result is an instance of the style (i.e., that it is correct) and that no rollback will be required
3 since concurrency was not investigated in the present work, the process is conducted on a single thread, even though every containers could be treated simultaneously.

Table 1 Experimentation results

	Containers	*Applications*	*Execution time (ms)*
Execution 1	2	4	850
Execution 2	3	10	1,531
Execution 3	3	50	12,655

6 Conclusions

This paper first underlines limitations of the currently available models of M2M systems. The ETSI is the only standardisation institute that provides a global end-to-end

view of M2M architectures. However, system adaptations are not addressed, even though M2M systems generally operate in highly evolving environments. To enable system management while still benefiting from functional advantages of the ETSI M2M standard, a multi-model approach is elaborated in this paper. A graph-based formalism allowing the definition of consistency-preserving reconfigurations has been targeted as a second model. More precisely, we formally characterised M2M systems by a graph grammar. Considering a bi-layered approach brings the issue of inter-layer coherency. Since the system is dynamic, layers have to evolve to fit its current state. Besides, evolutions can originate from both layers, since discovery and reconfigurations are conducted respectively in the functional and formal one. Consequently, we defined bi-directional communications. These lasts trigger actions updating a layer, such as both representations evolve in a consistent fashion.

Additionally, the appropriateness of the proposed framework is shown by defining high-level self-management policies. Noticeably, we elaborated scenarios related to M2M issues and procedures to cope with new requirements and/or prevent failures. Applicability of the approach is demonstrated through a smart-metering use case. In particular, we conducted a procedure preventing data loss comporting optimisation concerns. Implementation of this scenario also served as a basis for the approach evaluation.

Since we demonstrated the approach applicability, outlooks are twofold: application extension and further optimisation. Firstly, we plan on pursuing our implementations efforts to furnish an autonomic framework handling as many events as possible. Second, while considering reconfiguration policies in reaction of events, we supposed that events are independent and occurring in sequence. To improve the scalability and performance, we are considering the use of concurrency and priority within events and reactions plans. In particular, we should grant the possibility to modify an ongoing reconfiguration in reaction to an event.

Acknowledgements

The work presented in this paper has been partially funded by the ANR in the context of the project SOP, ANR-11-INFR-001.

References

Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J. and Sifakis, J. (2010) 'Automated conflict-free distributed implementation of component-based models', *International Symposium on Industrial Embedded Systems (SIES)*, pp.108–117.

Bradbury, J.S., Cordy, J.R., Dingel, J. and Werlinger, M. (2004) 'A survey of self-management in dynamic software architecture specifications', *Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed System (WOSS)*, pp.28–33, ACM, New York, USA.

Bruni, R., Bucchiarone, A., Gnesi, S., Hirsch, D. and Lafuente, A.L. (2008) 'Graph based design and analysis of dynamic software architectures', in: P. Degano, R. Nicola, and J. Meseguer (Eds.): *Concurrency, Graphs and Models*, Vol. 5065 of *Lecture Notes in Computer Science*, pp.37–56, Springer, Berlin/Heidelberg.

Chomsky, N. (1956) 'Three models for the description of language, information theory', in *IEEE Transactions on*, Vol. 2, No. 3, pp.113–124.

Ehrig, H. (1987) 'Tutorial introduction to the algebraic approach of graph grammars', in H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld (Eds.): *Graph-Grammars and Their Application to Computer Science*, Vol. 291 of *Lecture Notes in Computer Science*, pp.1–14, Springer, Berlin/Heidelberg.

ESNA: Energy Services Network Association [online] http://www.esna.org/.

ETSI FA TR: *ETSI Functional Architecture Technical Report* [online] http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/01.01.01_60/ts_102690v010101p.pdf.

ETSI TR: ETSI Technical Reports [online] http://www.etsi.org/technologies-clusters/technologies/m2m.

ETSI: European Telecommunications Standards Institute [online] http://www.etsi.org/Website/Technologies/M2M.aspx.

GMTE: Graph Matching and Transformation Engine [online] http://homepages.laas.fr/khalil/GMTE/.

Hirsch, D., Inverardi, P. and Montanari, U. (1999) 'Modeling software architectures and styles with graph grammars and constraint solving', in P. Donohoe (Ed.): *Software Architecture (TC2 1st Working IFIP Conf. on Software Architecture (WICSA1)*, pp.127–143, Kluwer, San Antonio, Texas, USA.

IPSO: IP-for the Connection of Smart Objects [online] http://www.ipso-alliance.org/.

Le Metayer, D. (1998) 'Describing software architecture styles using graph grammars', *IEEE Trans. Softw. Eng.*, Vol. 24, No. 7, pp.521–533.

Loulou, I., Kacem, A.H., Jmaiel, M. and Drira, K. (2004) 'Towards a unified graph-based framework for dynamic component-based architectures description in z', *Proceedings of the ICPS Internal Conference on Pervasive Services*, pp.227–234.

OMG, UML (2005) *Object Management Group, Unified Modelling Language Specification 2.0: Superstructure*, OMG doc. formal/05-07-04.

Pandey, S., Mup, M-S., C, M-H. and Hong, J.W. (2011) 'Towards management of machine to machine networks', in *Network operations and Management Symposium (APNOMS), 13th Asia-Pacific*, pp.1–7.

Roh, S., Kim, K. and Jeon, T. (2004) 'Architecture modelling language based on uml2.0', *Proceedings of the 11th Asia-Pacific Software Engineering Conference, APSEC '04*, pp.663–669, IEEE Computer Society, Washington, DC, USA.

Rozenberg, G. (Ed.) (1997) *Handbook of Graph Grammars and Computing by Graph Transformations*, Foundations, World Scientific, Singapore, Vol. 1.

Selonen, P. and Xu, J. (2003) 'Validating UML models against architectural profiles', *SIGSOFT Softw. Eng. Notes*, Vol. 28, pp.58–67.

Sharrock, R., Monteil, T., Stolf, P., Hagimont, D. and Broto, L. (2008) 'Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements', in *IJARAS*, Vol. 2, No. 1, pp.58–76.

TIA: Telecommunications Industry Association [online] http://tiaonline.org/standards.

Wermelinger, M. and Fiadeiro, J.L. (2002) 'A graph transformation approach to software architecture reconfiguration', *Science of Computer Programming*, Vol. 44, No. 2, pp.133–155.