FRAMESELF: A generic autonomic framework for self-management of distributed systems -Application on the self-configuration of M2M architecture using semantic and ontology

Mahdi Ben Alaya, Thierry Monteil

FRAMESELF: A generic autonomic framework for self-management of distributed systems - Application on the self-configuration of M2M architecture using semantic and ontology

Mahdi BEN ALAYA[1,2] , Thierry MONTEIL[1,2]
[1]CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France
[2]Univ de Toulouse, INSA, LAAS, F-31000 Toulouse, France
{ben.alaya, monteil}@laas.fr

Abstract— **M2M systems need to connect thousands of various fix and mobile machines that are widely distributed and evolve frequently according to their profile and context changes. The increasing complexity of managing current distributed systems needs new solutions. Designing autonomic systems, which are self-managing, and context aware is a solution and also a challenge. This paper proposes the FRAMESELF framework, a generic autonomic architecture based on decision models. The proposed solution capabilities are used for the self-deployment and self-configuration of machine-to-machine (M2M) systems. Components diagrams are illustrated to describe FRAMESELF modules and to show how they are connected together. FRAMESELF implements Devices Profile for Web Services (DPWS) protocol to describe and to discover managed resources. It implements multi models representation based on ontologies and graphs to describe the M2M concepts and relationships on a multi-level knowledge base. Two communication patterns modules based on service-oriented and event-driven communications are dynamically selected and configured in deployment plans. Finally, a smart metering scenario is experimented to validate this approach and to calculate the overload that FRAMESELF generates facing scalability.**

Keywords- autonomic computing, self-deployment, self-configuration, M2M architecture , ontology, graph.

I. INTRODUCTION

Over the last years, Internet of Things (IoT) has evolved at an exceptional speed. Things can both correspond to physical things (sensors, actuators, smartphones, machines, etc.) or immaterial ones (applications, multimedia content providers, directories, etc.). Such environments consist of a high amount of heterogeneous entities having mutual interactions and delivering high-level services that could be discovered, monitored, composed, and executed. One of the main concepts behind IoT is Machine-to-Machine (M2M) communication. M2M provides seamless integration of the heterogeneous participating machine domains overcoming the interoperability issues raised between them. The growing number of interconnected machines and the diversity of communication technologies and protocols as well as the increasing volume of exchanged data mirrored the increase of complexity in M2M systems. Deployment, configuration and maintenance of M2M platforms are costly in term of time and money and require a permanent presence of high skilled administrator. For this purpose, it is paramount to provide efficient M2M systems with capability of self-management to increase the autonomic potential of M2M systems.

On the first hand, autonomic computing paradigm was introduced by IBM to deal with system complexity which is inspired by the human autonomic nervous system that handles complexity and uncertainties, and aims at realizing computing systems and applications capable of managing themselves with minimum human intervention. In theory, Autonomic computing encompasses four self-management capabilities including self-configuring, self-healing, self-optimizing and self-protecting. Up until the present time, available autonomic systems are restricted to specific problems and are not applied in M2M systems due to the vertical fragmentation of M2M application domains. On the other hand, ETSI defined

a horizontal M2M service platform for M2M services interoperability that can be applied in different vertical domains. However, this M2M platform did not implements any self-management capabilities.

With the advent of technology, the possibility has arisen to integrate the autonomic computing paradigm into a horizontal M2M service platform. An autonomic M2M system should be able to automatically manage communication between interacting machines by taking actions based on the available information and the knowledge about what is happening in the environment. Administrators interact with the autonomic tools only to monitor business processes or alter the objectives. Overcoming complexity within M2M networks needs also a multi-view autonomic framework with semantic knowledge representation and reasoning capabilities. These kinds of autonomic manager should be capable of providing a semantic description of the interacting objects with respect to both their context characteristics and their behaviors or situations. From a concrete point of view, interacting entities need to be made semantically and automatically interoperable. Any management system will be, then, capable to dynamically detect these entities and capture their contexts in order to identify what are the changes that are happening in the environment, infer the current situation and react accordingly.

In the present effort, a generic autonomic framework called FRAMESELF [1] was designed and implemented for self-deployment and self-configuration of M2M communication services according to machines and applications description and environment changes. The substantive focus is the dynamic deployment of asynchronous event driven communications modules required to monitor distributed resources and to receive events as they happens from sensors, as well as the dynamic deployment of synchronous service oriented communications modules required to remote control distributed resources and invoke timely and accurate operations on actuators. FRAMESELF is designed in a way to be generic and extensible in order to be easily applied in various M2M domains. The proposed solution supports multi-models knowledge based on ontologies and graphs describing the ETSI M2M highlevel architecture, the communication service models, and the deployment graph plan.

The remainder of the paper is organized as follows: in section 2, a state of the art of the autonomic computing paradigm, knowledge model representation, ETSI M2M high level architecture, and existing communications patterns will be presented. In section 3, FRAMESELF architecture overview, UML components diagrams and knowledge models used for monitoring, analyzing, planning and execution will be detailed. In section 4, a smart metering use case will be demonstrated, and experiments results testing FRAMESELF scalability will be discussed. In section 5, a conclusion will be provided, and some perspectives and limitations will be pointed out.

II. STATE OF THE ART

A. *Autonomic computing paradigm*

Autonomic Computing is a paradigm proposed by IBM in 2001 [2,3,4]. It aims at developing distributed system capable of self-management to hide intrinsic complexity to administrators and users. An autonomic manager is organized into four main modules, which are Monitor, Analyzer, Planer and Executor. These modules share a same knowledge (managed resources details, policies, symptoms, request for change, plans, etc.), exploit policies based on goal and environment awareness and constitute together the MAPE-K control loop. Recent works on autonomic computing addressed some of the self-management capabilities applied in specific domains such as software deployment, data store, resource allocation, communication patterns adaptation, and query processing.

DeployWare [5] is an extensible component-based framework for automatic deployment of distributed and heterogeneous software systems. This approach defines three roles in the management of the software: the expert software, the system administrator and the end user. This framework proposes specific language for deployment (DSL Domain Specific Language) and a virtual machine for this language. The language is defined by a meta-model and provides a graphical notation in the form of a UML profile. Two concepts are important: the importance of defining roles and use of specific language comprehensible for users.

Gryphon [6] is an IBM project focusing on the design and development of highly scalable, available and secure publish/subscribe systems. It allows to dynamically adding brokers into the network to provide support for additional clients. It is able to respond to the failure of one broker in a network by rerouting traffic around the failed broker.

Astrolabe [7] is a hierarchically organized pear-to-pear query processing system. It support more expressive query used to collect the states of a very large-scale nodes according to zones. It also enables a variety of communication paradigms such as publish/subscribe, caching and multi-casting. The main drawback of astrolabe is that topology must be manually maintained by an administrator.

Facus [8] is a Framework for Adaptive Collaborative Ubiquitous Systems. This solution proposes a multi-layer modeling of system architectures in order to manage collaborative activities carried out by groups of user in complex issues. This framework proposes a Generic Collaboration Ontology (GCO) [9] and aims to deploy and configure collaborative session for users using event based communication modules. It is implemented using ontology and graph models and rule-oriented techniques. Although this framework uses generic collaboration ontology, there is no clear separation between the autonomic manager and knowledge base. Moreover, managed resources must generate the ontology instance, which induces a bad strong coupling between them and the framework.

Cited autonomic frameworks are designed to handle problems in specific domains, and are in most cases highly dependent on the type of managed resources. Although these solutions are effective in their domain and are relatively easy to use, but when it comes to maintain or to extend their architecture, they prove to be complex. These solutions are not modular and don't support multi-model representations in their knowledge bases for advanced management.

B. *Knowledge model representation*

Independently from the technical issues that may be encountered during the design of an autonomic system for M2M, a fundamental problem to be solved consists in the choice of the sort of formal vocabulary to enable representing vertical M2M domains knowledge in a horizontal way.

Relational database model is based on first-order predicate logic and enables to organize a collection of data items as a set of formally described tables from which data can be accessed easily. The purpose of this model is to provide a declarative method for specifying data and queries. Database enables to save only knowledge of a closed world system making difficult to extend or to share the relational model. It is more advantageous to represent knowledge according to the open world assumption [10] with machine understandable semantics.

Ontology model has proven beneficial for intelligent information integration, information retrieval, and knowledge management. Ontologies enable to index resources content using semantic annotations that can result in the representation of explicit knowledge that cannot be assessed and managed because of their mess. Given the increasing use of ontologies as a way of cleverly structuring a domain making use of semantic hierarchical and property/value relationships, utilizing a vocabulary of concepts/instances in order to describe rules, ontology represents now a very popular approach and is very useful to overcome challenges fixed in the proposed study [11].

The most popular language in the domain of semantic knowledge modeling making use of ontologies is the Web Ontology Language (OWL)[12]. OWL is a semantic an expressive schema language for publishing and sharing ontologies using RDF (the Resource Description Framework) extensions. OWL facilitates interoperability between entities by providing a shared understanding of the domain in question. It is an effective means for explicating implicit design decisions and underlying assumptions at system build time based on powerful deductive reasoning capabilities such as the Semantic Web Rule Language (SWRL) [13].

C. *M2M high level architecture*

M2M is a recent domain and standards are under construction. Even, the European Telecommunications Standards Institute (ETSI) is developing standards for M2M, the autonomic computing paradigm is missing to build self-managed M2M communication. In October 2011 ETSI published the M2M functional architecture standard [14]. Figure 1 describes the high level architecture for M2M as defined by ETSI specification. The architecture includes a Device and Gateway Domain, and a Network domain.

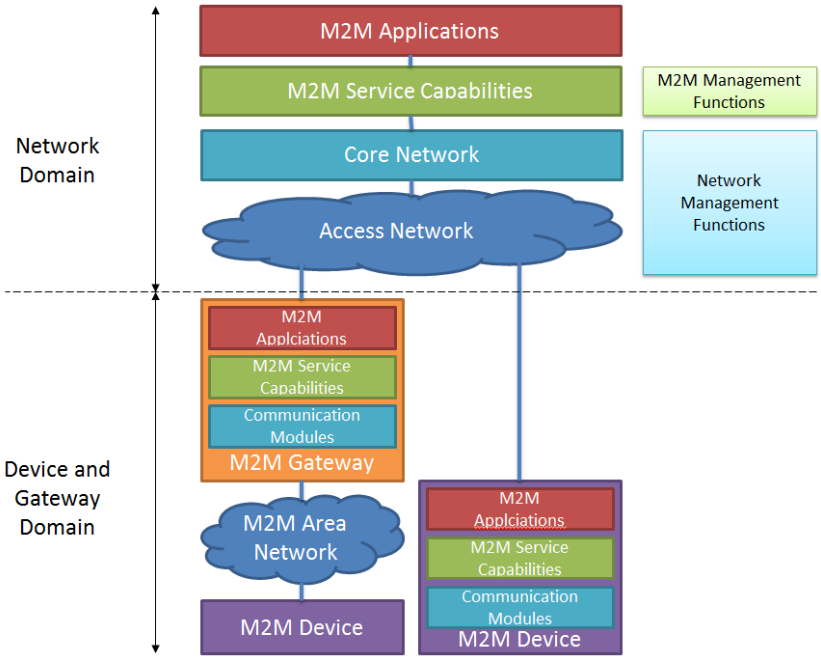

Figure 1. High level architecture for M2M

An M2M Device is able to run M2M Applications using M2M Service Capabilities. It can connect directly to the Network Domain via the Access network and may provide service to other devices connected to it that are hidden from the Network Domain. It can also connect to the Network Domain via an M2M Gateway through the M2M Area Network. M2M Area Network provides connectivity between M2M Devices and M2M Gateways. M2M Gateway also runs M2M Applications using M2M Service Capabilities, and acts as a proxy between M2M Devices and the Network Domain and may provide service to other devices connected to it that are hidden from the Network Domain.

Access Network allows the M2M Devices and Gateways to communicate with the Core Network. M2MService Capabilities Layer (SCL) provides M2M function that can be shared by different Applications. It provides services through a set of open interfaces to manage subscriptions and notifications pertaining to events. M2M applications run the service logic and use M2M Service Capabilities. Network Management Functions consist of all the functions required to manage the Access and Core networks: these include Provisioning, Supervision, Fault Management, etc. M2M Management Functions consists of all the functions required to manage M2M Service Capabilities in the Network Domain.

D. *Communication patterns*

The service oriented architecture (SOA) [15] relies on the request/response pattern underlying structure supporting communications between services. SOA defines how computing entities interact in such a way to enable one entity to perform a unit of work on behalf of another entity. The goal of SOA can be described as bringing the benefits of loose coupling and encapsulation to integration at highly

distributed system. SOA is quite useful for the web-services to synchronously request distributed actuators, but it is no suitable for decoupling asynchronous monitoring activities. In contrast to SOA, the event-driven architecture (EDA)[16] provides full decoupled communication between computing entities. EDA supports asynchronous publish/subscribe pattern. It prescribes that communication between components has to be performed on the basis of event notifications, where events are basically understood as changes in the state of something relevant in the system. EDA complements SOA by introducing long-running processing capabilities. Event consumers receive events as they happen, and loosely coupled services can be invoked to provide more timely and accurate data to customers. This kind of mixed communication architecture is called Event-Driven Service Oriented Architecture (EDSOA)[17].

III. AUTONOMIC FRAMEWORK PRINCIPLES

In this section, an overview of the FRAMESELF architecture will be presented. The global functioning of its main modules and the capability of the proposed solution to mix different models to manage system will be explained.

A. *FRAMESELF architecture overview*

The FRAMESELF architecture is based on the IBM autonomic architecture reference [2]. In our approach, the monitor, analyzer, planer, and executer operate as expert systems to emulate the decision-making ability of human experts. These modules are designed to solve complex problems by reasoning about knowledge, like an expert. Each module is divided into two parts, one fixed, independent of the system: the inference engine, and one variable: the knowledge base model. The FRAMESELF global architecture is described in Figure 2.

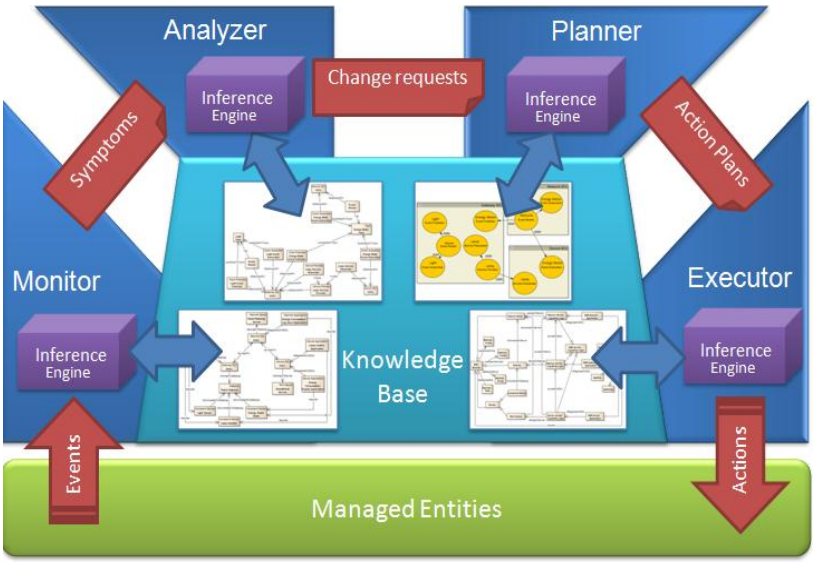

Figure 2. FRAMESELF architecture overview

The monitor role is to answer to the "what is happening?" question. It collects events from sensors from managed resources, updates a model describing the sensors environment and topology located on the knowledge base with relevant information from events, and infers new knowledge about symptom occurrences, then extracts relevant information and sends them to the analyzer.

The analyzer focuses on the "what to do?" question. It provides the mechanisms that correlate and model complex situation. These mechanisms allow the autonomic manager to learn about the environment and help to predict environment changes. The analyzer receives symptoms as input, uses

them to update a knowledge model describing the complex situation. The inference engine generates new knowledge about required requests for change (RFCs), and sends them to the planner.

The planner acts as a decision module and focuses on the question "how to do?". It saves received RFCs as goal states, reads models of possible actions and facts from the knowledge base and checks policies to guide its work. Then, it selects actions leading to the goal states and sends them to the executer.

The executor receives as input logical description of the sequence of actions to be executed, and consults a model containing actuators description and available operations details. It matches actions with their correspondent concrete operations, then performs the plan using actuators and controls the sequence of actions execution with consideration for dynamic updates. The executor must answer to the question "how is it done?" by generating reports and saving relevant information into the knowledge base.

B. *Monitoring process*

In this subsection, the monitor process will be explained in more details. The monitor component diagram and the M2M architecture ontology model will be presented.

1) *Monitoring component diagram*

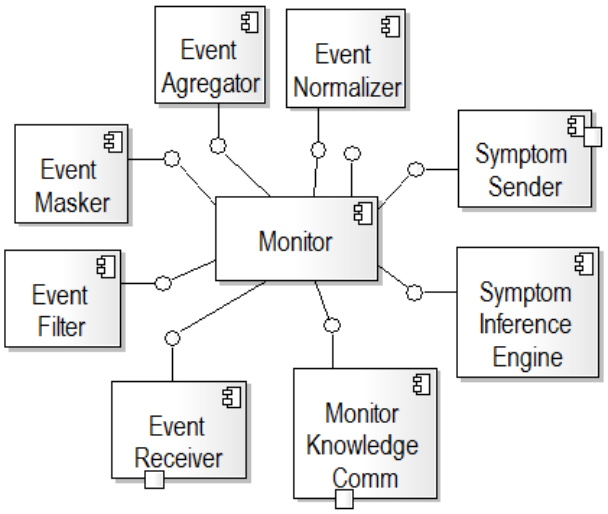

Figure 3. Monitor components diagram

Figure 3 describes the UML components diagram of the monitor:

- Event Receiver enables collecting events from different sources. It could be an event subscriber able to receive event from an event broker, it could be also an event log file parser, or an event service requester.
- Event normalizer consists in transforming, extending and formatting captured events in a standardized event format to make them consistent for processing.
- Event filter consists in discarding events that are deemed to be irrelevant for the management platform.
- Event aggregation consists in merging duplicates of the same event that may be caused by network instability.
- Event masking consists in ignoring events pertaining to systems that are downstream of a failed resource. Masking is different from filtering because masking allows to dynamically hiding unfiltered events according to the context changes.
- Monitor Knowledge Communication enables the monitor to read/write information from/to the M2M architecture and to read rules from the knowledge base.

- Symptom inference engine receives events as input, checks knowledge model, then generates symptoms as a response.
- Symptom sender enables delivering generated symptoms in different ways to the Analyzer using different technologies.

2) M2M Architecture knowledge model for monitoring

The model used by the monitoring component is based on ontology to manage the events during the discovery, configuration and deployment phases of the different part of the M2M system: M2M machines, M2M applications and M2M services. The monitor builds progressively an M2M ontology instance model, located the knowledge base, in line with the ETSI M2M high level architecture using received events.

The proposed ontology model, described in figure 3, is composed of four main classes: M2M Object, M2M Machine, M2M Application, and M2M Service Capability Layer (SCL).

- M2M Object class can be specialized to a hardware M2M machine or software M2M Application. An M2M Object can belong to a group, have a role, and have a specific profile.
- M2M Machine class can be described using machine name, type, location, state, and category details as well as runtime changing information such as the battery level, CPU, RAM, etc. A M2M Machine can be specialized to a Network Server, or a Gateway as well as a Sophisticated Device or Constraint Device.
- M2M SCL class belongs to a M2M machine and runs deployed communication services. It can be specialized to a Network SCL, a Gateway SCL, or a Device SCL.
- M2M Application class can be described using application type, targeted location, and targeted category information, and is able to register to a SCL. A M2M Application can be specialized to a Network Application, a Gateway Application, or a Device Application.

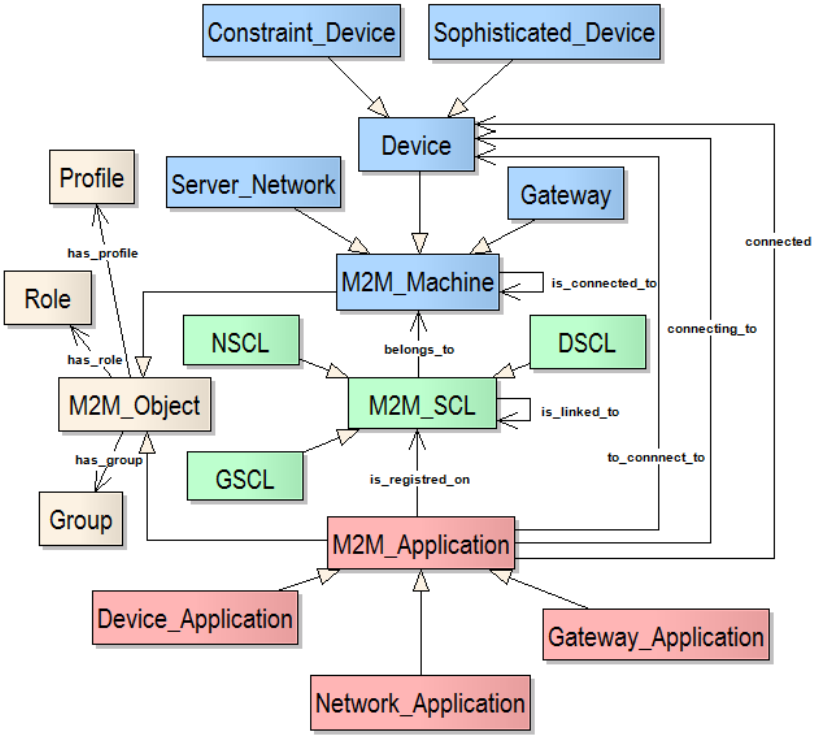

Figure 4. M2M architecture ontology model

3) Example of detecting non configured M2M constraint devices and M2M applications using SWRL rules.

This model can be used to interconnect M2M application with their corresponding M2M constraint devices. The constraint device could be a sensor or an actuator according to its type information. It could own one communication service and have one of the following status: “not_configured”, “to_configure”, “configuring”, and “configured” to enable handing constraint device life cycle. An M2M application could interact with one or more constraint devices according to its targeted location, and targeted category information. The following types are used to handle connections life cycle: “not_connected_to”, “to_connect_to”, and “connected_to”.

As a first step, the monitor applies a semantic rule on the ontology model instance to detect non configured devices and toggle their status to: “to_configure”. The following SWRL rule is used for this issue:

$$\text{Constraint\_Device}(?cd) \wedge \text{has\_status}(?cd, \text{"not\_configured"}) \rightarrow \text{has\_status}(?cd, \text{"to\_configure"})$$

As a second step, the monitor applies a second rule to detect eventual correlation between existing applications and available constraint devices to spot missing connections between them. If a constraint device location is equal to an application targeted location, and if a constraint device type is equal to an application targeted type, then the monitor concludes that this application should be connected to this constraint device. The following SWRL rule is used for this issue:

$$\text{Constraint\_Device}(?cd) \wedge \text{has\_location}(?cd, ?l) \wedge \text{has\_category}(?cd, ?c) \wedge \text{Application}(?a) \wedge \text{has\_target\_location}(?a, ?tl) \wedge \text{has\_target\_category}(?a, ?tc) \wedge \text{swrlb:equal}(?l, ?tl) \wedge \text{swrlb:equal}(?c, ?tc) \rightarrow \text{to\_connect\_to}(?a, ?cd)$$

After executing these rules and inferring new axioms, the monitor generates symptom containing details of constraint devices to be installed and applications to be connected, and sends them to the analyzer.

C. *Analyzing process*

In this subsection, the analyzer components diagram and M2M communication services knowledge model will be detailed.

1) Analyzing components diagram

Figure 5 describes the UML components diagram of the analyzer:

- Policy Validator enables to validate a policy entered by the administrator, located on the knowledge base, to guide the control loop operations.
- Symptom Receiver enables collecting events from the monitor sources using different technologies.
- Symptom Normalizer consists in formatting received symptom in a standardized symptom format to make them consistent for analyzing.
- Analyzer Knowledge Communication enables to handle the M2M communication services model and read rules and policies from the knowledge base.

- Request Inference Engine receives Symptom as input, checks knowledge model, then generates Request for change as a response.
- Request Sender enables to send generated request for change to the planner or another module using different technologies.

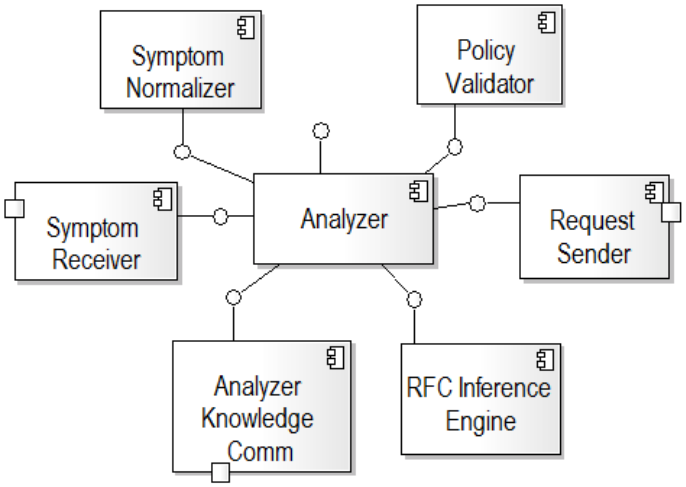

Figure 5. Analyzer components diagram

2) Communication services knowledge model for analyser

Based on received symptoms, the analyzer tries to determine required communication components that can be installed on M2M constraint devices and M2M applications. To perform this task, a service communication knowledge model containing details of relevant communication patterns and information of how this patterns work is required. In this study, the event-driven and service-oriented communication patterns are considered as shown in figure 6.

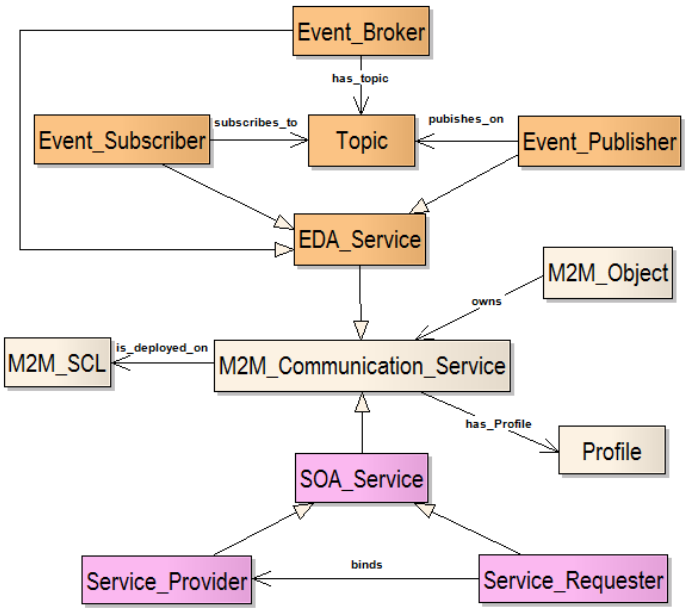

Figure 6. M2M communication services ontology model

A M2M communication Service is defined using a profile. It can be deployed on a M2M SCL and can be owned by an M2M object. According to the request/response communication pattern, a SOA service

could be specialized to Service Provider, or Service Requester. A service requester invokes a service provider to execute an action and gets back a response. According to the publish/subscribe communication pattern, an EDA service could be specialized to an event publisher, an event subscriber, or an event Broker. The event broker has a list of topics and the publisher publishes events on a topic. A subscriber is linked to one or more topics, and only receives events that are of interest.

3) Example of defining communication services for M2M constraint devices and M2M applications using SWRL rules

Firstly, the analyzer applies semantics rules on the ontology model instance to detect appropriate communication service for not installed constraint devices. A constraint device doesn't implement a SCL, so communication services will be deployed on the gateway SCL to which it is connected. If a constraint device has the "actuator" type, then a service provider will be selected and deployed for this actuator, and the constraint device status will be changed to "configuring". The following SWRL rule is applied for this issue:

Constraint_Device(?cd) ∧ has_status(?cd, "to_configure") ∧ is_connected_to(?cd, ?g) ∧ belongs_to(?gscl1, ?g) ∧ has_type(?cd, "actuator") ∧ swrlx:makeOWLIndividual(?sp, ?cd) → Service_Provider(?sp) ∧ owns(?cd, ?sp) ∧ is_deployed_on(?sp, ?gscl1) ∧ has_status(?cd, "installing")

If a constraint device has the "sensor" type, then an event publisher will be selected and deployed for this sensor, and the constraint device status will be changed to "configuring". The following SWRL rule is applied for this issue:

Constraint_Device(?cd) ∧ has_status(?cd, "to_configure") ∧ is_connected_to(?cd, ?g) ∧ belongs_to(?scl1, ?g) ∧ has_type(?cd, "sensor") ∧ has_category(?cd, ?c) ∧ Event_Broker(?eb) ∧ is_deployed_on(?eb, ?scl1) ∧ swrlx:makeOWLIndividual(?ep, ?cd) ∧ swrlx:makeOWLIndividual(?to, ?cd) → Event_Publisher(?ep) ∧ Topic(?to) ∧ has_name(?to, ?c) ∧ is_deployed_on(?ep, ?scl1) ∧ has_topic(?eb, ?to) ∧ publishes_on(?ep, ?to) ∧ owns(?cd, ?ep) ∧ has_status(?cd, "installing")

Secondly, the analyzer applies semantics rules to determine appropriate communication services for M2M applications. If an application must be connected to an actuator, then a service requester will be selected to be deployed on the SCL to which the application is registered. The application service

requester binds to the actuator service provider. The link between the application and the actuator is changed to "connecting_to". The following SWRL rule is applied for this issue.

$$\begin{aligned}&\text{Application}(?a) \wedge \text{is\_registred\_to}(?a, ?scl2) \wedge \text{Constraint\_Device}(?cd) \wedge \text{to\_connect\_to}(?a, ?cd) \wedge \\ &\text{has\_type}(?cd, \text{"actuator"}) \wedge \\ &\text{swrlx:makeOWLIndividual}(?sr, ?a) \wedge \\ &\text{Service\_Provider}(?sp) \wedge \text{owns}(?cd, ?sp) \\ &\rightarrow \text{Service\_Requester}(?sr) \wedge \\ &\text{is\_deployed\_on}(?sr, ?scl2) \wedge \\ &\text{binds}(?sr, ?sp) \wedge \text{owns}(?a, ?sr) \wedge \text{connecting}(?a, ?cd)\end{aligned}$$

If an application must be connected to a sensor, then an event subscriber will be selected and deployed on the SCL to which the application is registered. This event subscriber subscribes to the same topic as the corresponding sensor event publisher. The link between the application and the actuator is changed to "connecting_to". The following SWRL rule is applied for this issue.

$$\begin{aligned}&\text{Application}(?a) \wedge \text{is\_registred\_to}(?a, ?scl1) \wedge \text{Constraint\_Device}(?cd) \wedge \text{has\_type}(?cd, \text{"sensor"}) \wedge \\ &\text{to\_connect\_to}(?a, ?cd) \wedge \text{Topic}(?to) \wedge \\ &\text{owns}(?cd, ?ep) \wedge \text{publishes\_on}(?ep, ?to) \wedge \\ &\text{swrlx:makeOWLIndividual}(?es, ?a) \\ &\rightarrow \text{Event\_Subscriber}(?es) \wedge \\ &\text{is\_deployed\_on}(?es, ?scl1) \wedge \text{subscribes\_to}(?es, ?to) \wedge \text{owns}(?a, ?es) \wedge \text{connecting}(?a, ?cd)\end{aligned}$$

After executing these rules and inferring new axioms, the analyzer generates requests for change containing information of M2M communication services to be deployed and how they are linked together, and sends them to the planner.

D. Planning process

In this subsection, the planer process will be explained in more details. The planner components diagram and the deployment graph model will be presented.

1) Planner components diagram

The UML components diagram of the planner is composed by (figure 7):

- Policy Interpreter enables to processes a policy located on the knowledge base and to apply it to guide the decisions that affect the autonomic manager behavior.
- RFC Receiver receives RFCs from the analyzer using different technologies.
- RFC Normalizer consists in formatting received request for change according to a RFC template to make them ready for planning.
- Planner Knowledge Communication enables to handle the deployment graph model and read policies from the knowledge base.

- Plan Generator receives RFCs as input, checks knowledge model, then generates action plans as response.
- Plan Sender enable to sends generated action plan to the Executer or another module using different technologies.

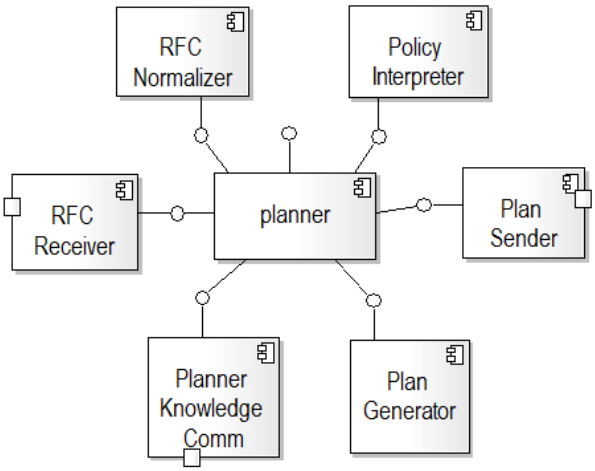

Figure 7. Planner components diagram

2) Deployment plan graph model.

To deploy requested communication services, the planner generates a deployment plan graph model according to received RFCs details. A 2-level nested graph model based on the GraphML [18] standard is proposed to model the deployment plans. In the first level graph, nodes represent M2M SCLs and contain information of M2M machines context. In the second level, nodes represent M2M communication services to be deployed on the SCLs and contain configuration parameters (topics name, service provider address, etc.) as well as deployment actions such as install, start, update, stop, uninstall, and configure. Arrows direction informs of the next services to deploy and arrows values inform of the sleeping time before moving onto the next operation. The second level graph is considered as a precedence graph to ensure deployment scheduling.

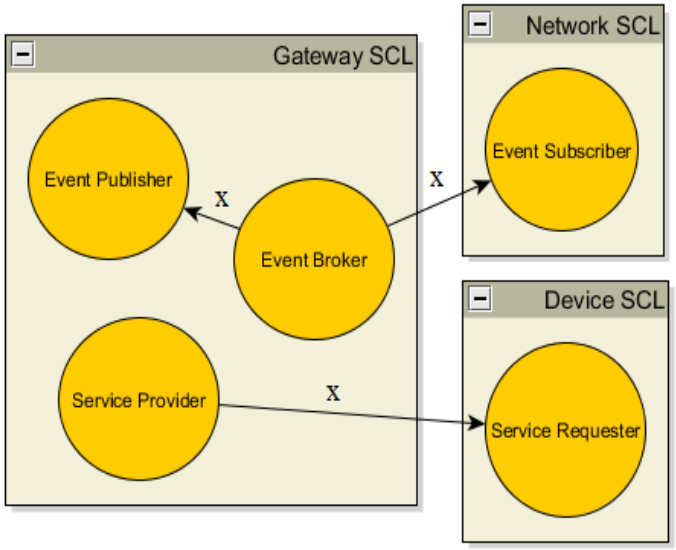

Figure 8. Deployment graph plan model

The following rules are used for this issue: For EDA communications, event brokers will be deployed before event publishers and event subscribers. For SOA communications service providers will be deployed before service requesters. An example of a graph plan model for communication service deployment is given in figure 8. According to this model instance, the first step consists of deploying an event broker and a service provider. The next step consists of deploying, after "X" seconds, an event subscriber, an event publisher, and a service requester.

E. *Executing process*

In this subsection, the Execution components diagram and the deployment graph model will be presented.

1) *Executor components diagram*

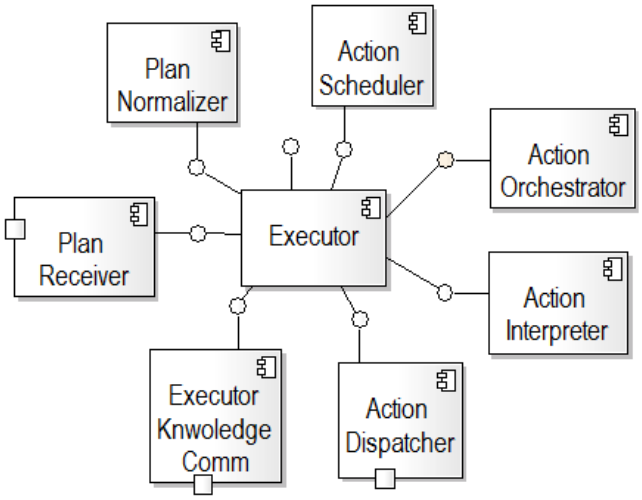

Figure 9. Executor components diagram

Figure 9 describes the UML components diagram of the planner:

- Plan Receiver enables receiving action plans from the planner using different technologies.
- Plan Normalizer consists in formatting received plan to make them easy to parse by the Action interpreter.
- Executor Knowledge Communication enables to handle the deployment result graph model from the knowledge base.
- Action Interpreter focuses on processing received actions and checking whether the executor has the rights and means to execute them.
- Action Orchestrator enables to describe the automated arrangement and coordination of actions in time.
- Action scheduler determines, for simultaneous actions, which action to be executed next to load balance the system.

2) *Deployment result report*

Initially, a M2M deployment service is started in each SCL that can be invoked by the executor to deploy new M2M communication services according to the deployment plan graph. After each operation, the executor generates a result reports containing deployment operations status (Successful, fail), new components life-cycle details (installed, resolved, starting, active, stopping, uninstalled, etc.) and eventual errors messages. If the deployment process fails, the executor is able to display an alert message with the

occurred errors to the administrator, to return back to the previous deployment checkpoint, or, if it is possible, to plan a new deployment graph taking in consideration deployment errors.

IV. EXPERIMENTS

In this section a smart metering use case will be demonstrated and performance results will be presented.

A. *Smart metering use case description*

Let's consider a M2M distributed system composed of six M2M machines connected as follows: a home gateway and a Smartphone connected to a smart metering server through a wide area network. Three constrained devices: a smart meter, a light sensor, and a lamp actuator connected to the home gateway through a home area network. These constraint devices will be used by four M2M applications as following: a home gateway application for automatic lamp regulation according to the light level, a M2M server application for energy consumption log saving, and two M2M applications for smartphone: the first one for lamp controlling and the second one for energy consumption displaying. Figure 10 describes the use case architecture.

To be able to manage this M2M system, the autonomic manager must be adapted for resource-constrained devices and supports standardized mechanisms for seamless interaction with constraint managed resources by meeting a set of requirements:

- Transport-neutral mechanisms to address managed resources hosted services,
- Discovery protocol to locate available services,
- Generic protocol for accessing service-based resources representation,
- Messaging protocol that allows services to accept subscriptions for event notifications,

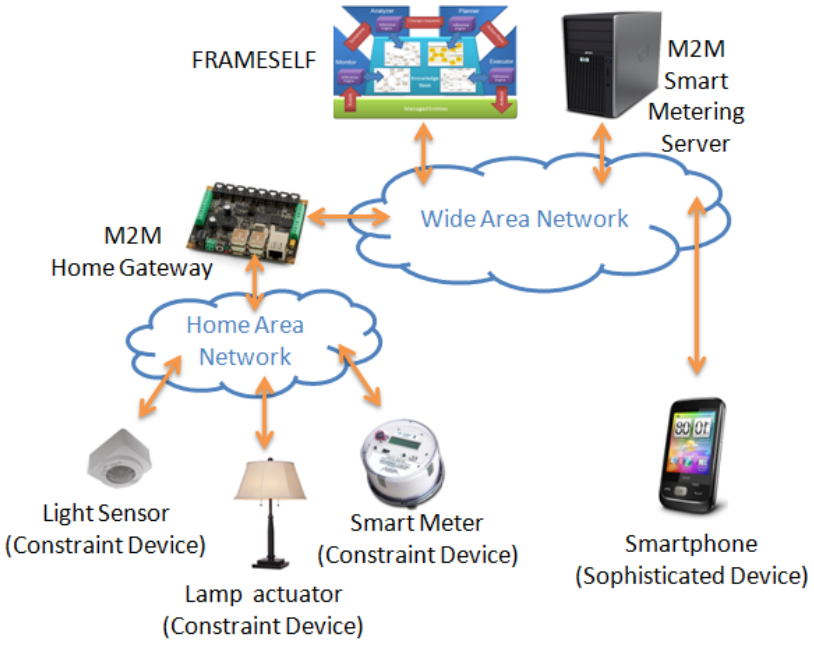

Figure 10. M2M smart use case architecture

B. *Seamless integration of managed resources using DPWS*

To meet requirements cited above, the Devices Profile for Web Services (DPWS) [19] specification was considered in the proposed approach. DPWS defines a minimal set of implementation constraints to enable secure web service messaging, discovery, description, and eventing that is fully aligned with web

services standards and includes numerous extension points for seamless integration of device-provided services in a highly distributed system. DWPS is based on existing Web Service (WS) standards such as WS-Addressing, WS-Discovery, WS-Eventing, WS-MetadataExchange, WS-Transfer, and WS-Security adapted for mobile and resource-constrained devices.

Using DPWS, a device is able to host and offer two types of services. The first one is the "Operation Service" which is suitable for making synchronous communication on actuators based on request/response design pattern. It is further used to declare and return operations or events input and output parameters as well as any faults that may occur on invocation. The second type is "Default Event Source Service" which is suitable for making asynchronous communication on sensors based on publish/subscribe pattern. The DPWS framework supports two types of WS-Eventing compliant events: notifications and solicit-response operations. While the first ones represent one-way messages sent from the event source to its subscribers, the later additionally includes response messages sent back from the subscribers to the source.

C. *Self-configuration using FRAMESELF*

Monitor discovers available M2M machines and M2M applications using WS-Discovery, and generates the M2M architecture ontology instance. After applying semantic rules using the JESS [20] inference engine, the Monitor detects that the three constraint devices should be configured:

- The lamp regulating application should be connected to the light sensor and the lamp actuator,
- The energy log saving application should be connected to the energy smart meter,
- The lamp controlling application and energy consumption displaying application should be connected respectively to the lamp actuator and the energy smart meter.

To sum up, three constraint device configurations and five application connections must be analyzed which represents eight symptoms to be generated and sent to the analyzer. The M2M architecture ontology instance is described in figure 11.

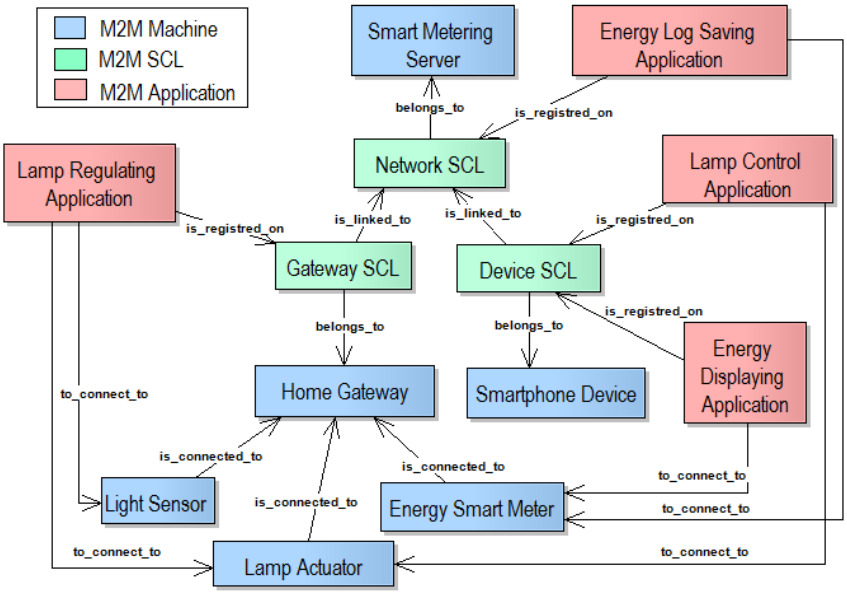

Figure 11. M2M architecture ontology instance

On the second step, the analyzer updates the communication service ontology instance based on the data extracted from the eight symptoms. After applying semantic rules using the JESS inference engine, the analyzer determines required M2M communication services:

- Light sensor owns an event publisher and publishes event on the light topic of the gateway broker.

- The gateway application owns a light event subscriber to collect events from the light topic.
- The gateway application checks the light intensity and then uses a lamp service requester to control the lamp service provider.
- The Smartphone application owns a lamp service requester to remote control the lamp service provider.
- The energy smart meter owns an event publisher and publishes events on the energy topic of the event broker.
- The smart metering server and the Smartphone each owns an energy metering subscriber and uses it to collect events for the energy topic to, respectively, save and display home energy consumption.

The M2M communication ontology instance is given on figure 12.

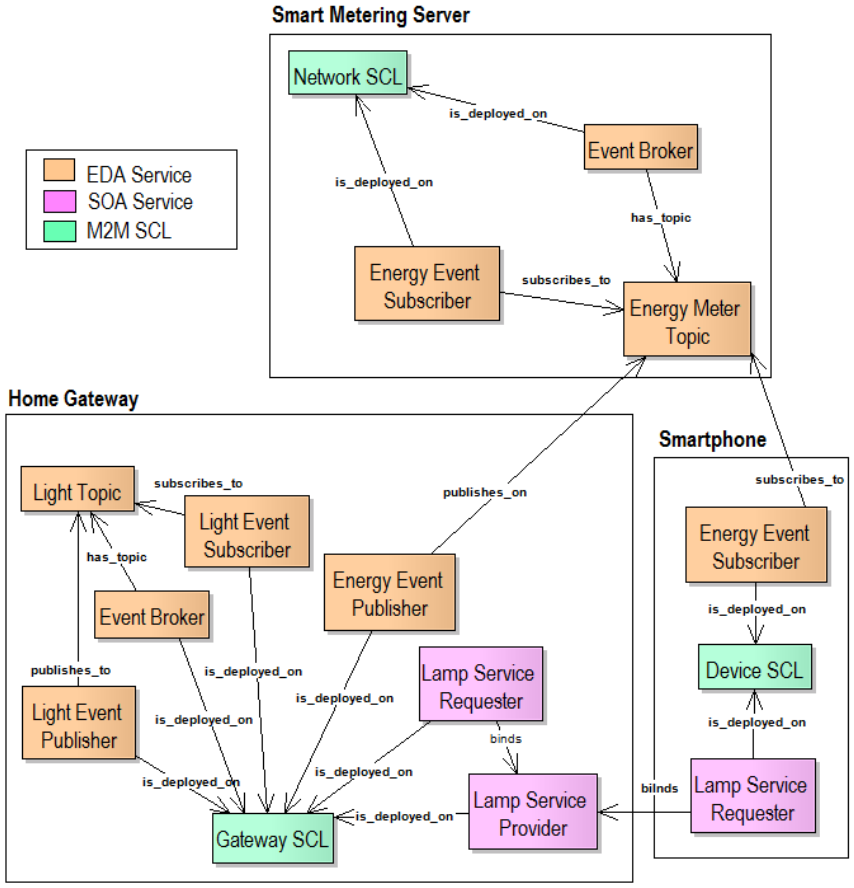

Figure 12. M2M communication services ontology instance

On the third step, the planner generates the deployment graph plan based on received RFCs. It plans the deployment of the event brokers of the home gateway and the smart metering server, and the lamp service provider. It plans the configuration of a light topic on the gateway broker and an energy topic on the smart metering server broker. The lamp service provider is configured to be invoked externally. The planner plans the deployment of the smart meter event publishers and the light sensor event subscribers and configures them to connect, respectively, to the energy and the light topics. The planner plans also the deployment of the lamp service requester and configures it to bind the lamp service provider. Figure 13 depicts the generated deployment graph plan instance.

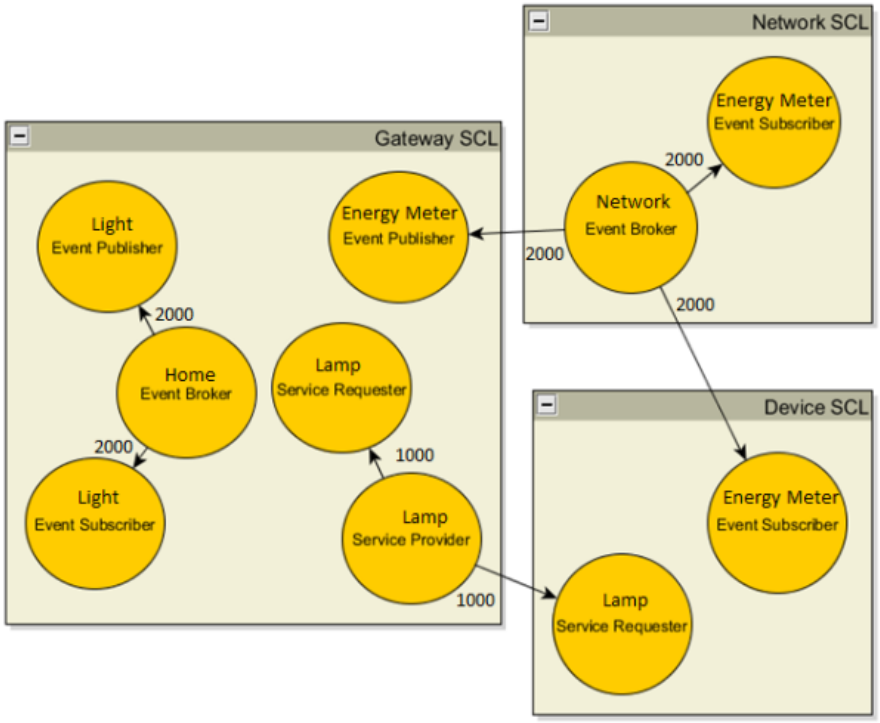

Figure 13. Deployment graph plan example

On the last step of the control loop, the executor reads M2M communication services description from the graph plan model, and determines the location of the corresponding software components. It finally deploys required software components on their appropriate M2M machines, configures and starts them according to the plan using the OSGi technology [21]. The executor generates a report detailing the result of each actions and the state of each deployed components.

D. *Performance results*

In this part performance results were conducted to calculate the overload generated by FRAMESELF during the control loop steps in the context of a big M2M system. The capacity of the autonomic loop iteration and the knowledge base model processing and reasoning overload will be discussed. The experimentations were performed using a computer having an Intel Core i7-2670QM CPU 2.20GHz and a 8GB of RAM.

1) *FRAMESELFcontrol loop performance*

The monitoring, analysing, planning and executing phases of the control loop are done by four threads. It looks like a pipeline. First measures are done to evaluate the processing capacity of the loop with no knowledge base only to evaluate the components architecture. Time is firstly collected at the beginning of a burst of events and secondly when all events have created a single action in the executor module. On figure 14, the processing capacity is shown for the treatment of a burst. For a single event, wake-up time of thread invocation of component and runtime of component need 30 ms. FRAMESELF could create action every 0.55 ms in a burst. In a pipeline system, it corresponds to the time of treatment of the slowest element of the pipeline. So in a burst, an event creates an action in less than 2.2 ms. The proposed components architecture is not a limitation for the capacity of FRAMESELF.

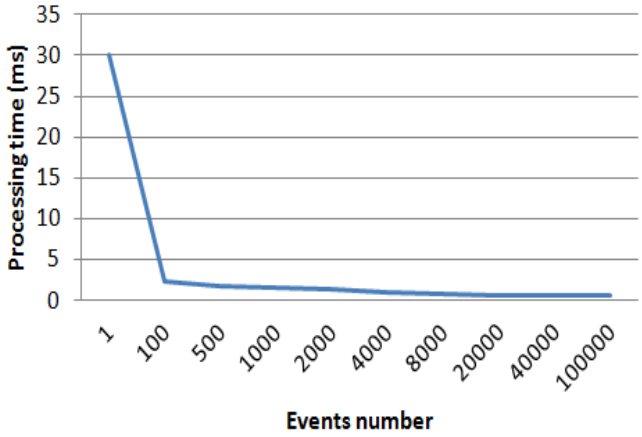

Figure 14. Processing capacity of the autonomic loop

2) Knowledge base processing and reasoning performance

Scalability could become a serious problem when dealing with OWL models containing a large number of entities and especially when using inference engine with complex rules. In fact, each time, the inference engine must load the full OWL instance content before reasoning, and save all inferred knowledge when finished. Therefore, scalability tests were focused on the monitoring and analyzing functions to measure the impact of using OWL with JESS inference engine on the scalability of the proposed system.

The curve presented in figure 15 describes the time needed for the monitor to detect simultaneously up to 90 events of non-configured devices. Although it is rare to reach this high number, because most of time device and application will be added one by one, the monitor reasoning time remains low.

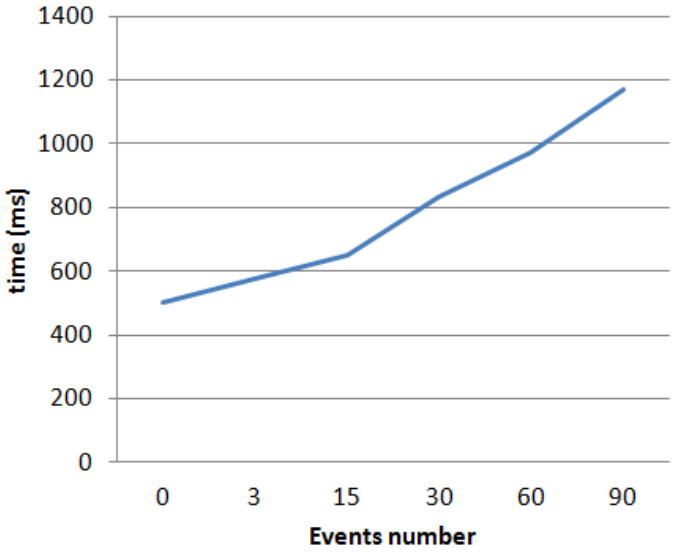

Figure 15. Monitoring capacity

The curve presented on figure 16 tells about the time needed for the analyzer to handle simultaneously up to 250 symptoms. As a reminder: each symptom contains details of a machine or an application to

configure. Beyond 200 symptoms, the overload increases considerably and may disrupt the normal functioning of the system.

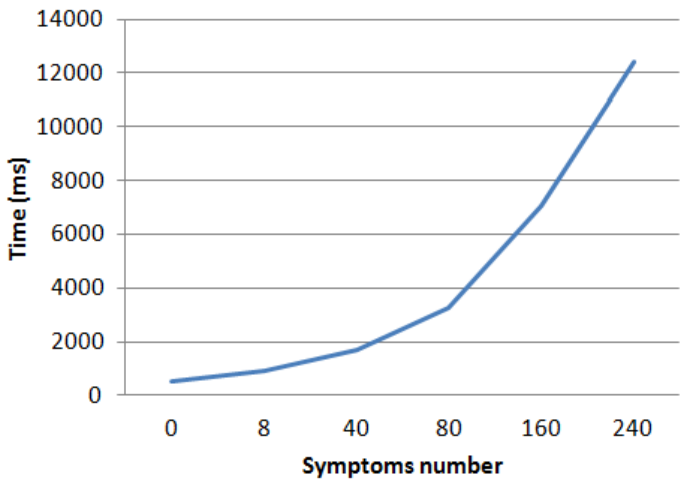

Figure 16. Analyzing capacity

V. CONCLUSION

In this paper, the basis of a generic autonomic framework called FRAMESELF aiming at self-managing of M2M system was presented. The FRAMESELF global architecture and the functioning of its main module were explained. The proposed solution focused on the self-configuration of M2M communication services according to semantic annotation collected from M2M objects within a M2M system. A multi-model knowledge based was considered to perform the control loop steps and a M2M architecture ontology model was defined for the monitoring process. A M2M communication services ontology model based on the publish/subscribe and request/response design patterns was specified for the analyzing process and a deployment graph plans was generated during the planning process. This approach was demonstrated using a M2M smart metering use case and performance results were presented to measure the overload generated by FRAMESELF facing scalability.

As future work, we propose to apply FRAMESELF for self-configuring M2M communication services in more complex M2M scenario by integrating standardized ontology model like the OWL-S [22] and the SSN [23] ontologies to represent in more details services, events and devices. We propose to such advanced models to support additional self-management capabilities such as self-optimization, self-healing and self-protecting and apply it to manage new M2M business cases such as e-health, domotic, and smart grid. We plan also to contribute on M2M standardization within ETSI for the autonomic computing and semantic aspects.

ACKNOWLEDGEMENT

This work is part of the European project ITEA2 A2NETS. It will be integrated on the smart metering business case demonstrator.

REFERENCES

1. Mahdi Ben Alaya, Thierry Monteil, "FRAMESELF: A Generic Context-Aware Autonomic Framework for Self-Management of Distributed Systems," *wetice, IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp.60-65, 2012

2. Parashar, Manish, and Salim Hariri. Autonomic Computing: Concepts, Infrastructure, and Applications. *Boca Raton: CRC/Taylor & Francis*, 2007. Print.
3. Kephart, J.O.; Chess, D.M.; , "The vision of autonomic computing," *Computer* , vol.36, no.1, pp. 41-50, Jan 2003 doi: 10.1109/MC.2003.1160055
4. S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the vision of autonomic computing. *Computer*, 43:35{41, 2010.
5. Flissi, A.; Dubus, J.; Dolet, N.; Merle, P.; , "Deploying on the Grid with DeployWare," *8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID '08*, vol., no., pp.177-184, 19-22 May 2008, doi: 10.1109/CCGRID.2008.59.
6. Peter R. Pietzuch and Sumeer Bhola. 2003. Congestion control in a reliable scalable message-oriented middleware. *In Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware '03),* Markus Endler (Ed.). Springer-Verlag New York, Inc., New York, NY, USA, 202-221.
7. Birman, K.P.; van Renesse, R.; Vogels, W.; "Scalable data fusion using Astrolabe", *Proceedings of the Fifth International Conference on Information Fusion, 2002*, vol.2, no., pp. 1434- 1441 vol.2, 2002, doi: DOI: 10.1109/ICIF.2002.1020984.
8. Sancho G., Villemur T., Tazi S., "An Ontology-driven Approach for Collaborative Ubiquitous Systems", InderScience Publishers Mai 2010. pp 263 - 279. *www.inderscience.com/ijac/ International Journal of Autonomic Computing (IJAC)* ISSN 1741-8577 - ISSN (Print): 1741-8569
9. Sancho, G., Tazi, S., Villemur, T.: A Semantic-driven Auto-adaptive Architecture for Collaborative Ubiquitous Systems. *In: 5th International Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST'2008)*, Cergy Pontoise (France) (2008) 650–655, doi: 10.1145/1456223.1456354.
10. Russell, Stuart J.; Norvig, Peter (2010). Artificial intelligence: A modern approach (3rd ed. ed.). *Upper Saddle River: Prentice Hall*. ISBN 9780136042594.
11. Stojanovic, L., Schneider, J., Maedche, A., Libischer, S., Studer, R., Lumpp, Th., Abecker, A., Breiter, G., Dinger, J., "The role of ontologies in autonomic computing systems," *IBM Systems Journal* , vol.43, no.3, pp.598-616, 2004
12. OWL Web Ontology Language Reference, Mike Dean and Guus Schreiber, Editors, *W3C Recommendation,* 10 February 2004.
13. W. Zhang and K. M. Hansen. Towards self-managed pervasive middleware using owl/swrl ontologies. *In Fifth International Workshop on Modeling and Reasoning in Context (MRC 2008)*, pages 1–12, Delft, TheNetherlands, Jun. 2008.W3C Member Submission 21 May 2004M.
14. Boswarthick, David, Omar Elloumi, and Olivier Hersent. M2M Communications: A Systems Approach. *Chichester, West Sussex, U.K.*: Wiley, 2012. Print.
15. Erl, Thomas. *SOA Design Patterns*. Upper Saddle River, NJ: Prentice Hall, 2009. Print.
16. Mühl, Gero, Ludger Fiege, and Peter Pietzuch. Distributed Event-based Systems. Berlin: *Springer, 2006*. Print.
17. Laliwala, Z., Chaudhary, S., "Event-driven Service-Oriented Architecture", *2008 International Conference on Service Systems and Service Management, vol.,* no., pp.1-6, June 30 -July 2 2008

18. U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M.S. Marshall: GraphML Progress Report: Structural Layer Proposal. *Proc. 9th Intl. Symp. Graph Drawing (GD '01),* LNCS 2265, pp. 501-512.
19. F. Jammes, A. Mensch, and H. Smit. Service-oriented device communications using the devices profile for web services. *In Proceedings of the 3rd international workshop on Middleware for* pervasive and ad-hoc computing, MPAC '05, pages 1-8, New York, NY,USA, 2005. ACM.
20. Choonhwa Lee, D. Nordstedt, and S. Helal. "Enabling Smart Spaces with OSGi." IEEE Pervasive Computing 2.3 (2003): 89-94. Print.
21. Friedman-Hall, Ernest (2003). "Jess in Action: Rule Based Systems in Java" . Pearson Education. Retrieved March 30, 2012. ISBN 1-930110-89-8
22. David Martin, Mark Burstein, Drew Mcdermott, Sheila Mcilraith, Massimo Paolucci, Katia Sycara, Deborah L. Mcguinness, Evren Sirin, and Naveen Srinivasan. 2007. Bringing Semantics to Web Services with OWL-S. *World Wide Web* 10, 3 (September 2007), 243-277. DOI=10.1007/s11280-007-0033-x http://dx.doi.org/10.1007/s11280-007-0033-x
23. Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, Kerry Taylor, The SSN ontology of the W3C semantic sensor network incubator group, Web Semantics: Science, Services and Agents on the World Wide Web, Volume 17, December 2012, Pages 25-32, ISSN 1570-8268, 10.1016/j.websem.2012.05.003.