

Adaptive architectural model for Future Internet applications

Marina Mongiello¹, Luigi Alfredo Grieco¹, Massimo Sciancalepore¹, and Elvis Vogli¹

¹Politecnico di Bari – Via E. Orabona, 4 – 70125 Bari, Italy
{firstname.lastname}@poliba.it

Abstract. Interoperability, flexibility and adaptability are key requirements of Future Internet applications. Convergence of contents, services, things and networks seems to be the cornerstone to fulfill these requirements. In this paper we propose a model for runtime composition of software applications in sensors networks based on data, processes and technology, in order to design an "on the fly" architecture of a software system. The model is graph-based and composed by two control levels: a formal model and the instantiation level. An algorithm extracts a subgraph that identifies the applications to be executed according to changes in the external context. The proposed approach has been instantiated in a use case example in a smart home environment, to evaluate the usefulness of the approach and the applicability of the model in actual scenarios.

Keywords: formal model, runtime architectural model, sensor networks

1 Introduction and motivation

Future Internet applications should be able to handle dynamic changes in user experience and interoperability between different technologies, data, and processes. Convergence of contents, services, things, and networks seems to be the relevant direction taken by these applications. [2], [4].

Such complex and composite source of data ranging from signals, raw data, events and complex events needs technological and theoretical formalization. In the light of all these novelties, adaptive mechanisms to develop and orchestrate services and applications are emerging [3], [5].

A formal approach for runtime composition of software applications in sensor networks is proposed hereby. The approach is made of two control levels: a technology independent level and an instantiation one. The first level catches different configurations of adaptive software modeled using a graph structure. Each node in the graph can be classified as a data or a process or a technology. A data node represents information derived from the external context (e.g., a data detected by a sensor). Data trigger processing of software applications, i.e. process node. Finally, technology node specify features of the devices (for example the state of the device where the application will be executed, the state

of the memory of the device or the state of the middleware used to make software application interact with each other). At the instantiation level, instead, the graph model is contextualized to application context that are platform dependent and belong for example to Java, Android, and any other technology environment.

Data processing and technologies selection and management is driven by specification of operational requirements. On the graph model, hence at the first control level, we also define an algorithm to extract a subgraph from the graph by minimizing a cost function. The algorithm finds the best sequences of (data, process, technology) terms minimizing the cost due to resources utilization. Such sequences of nodes and the related paths correspond to the selected orchestration of services or application in response to the context behavior. The main advantage of the proposed approach is to possibility to delay architectural decisions at run-time and to build the architecture on the fly, depending on the specified requirements.

The proposed approach has been instantiated in a use case example in a smart home environment, to evaluate the usefulness of the approach and the applicability of the model in realistic scenarios.

The remaining of this paper is organized as follows. Section 2 introduces background information needed for clarifying the proposed approach. The proposed formal model is defined in Section 3. Instantiation of the model with respect to a realistic use case scenario is described in Section 4. Conclusions and future works close the paper.

2 Background

In this section we introduce techniques and notions that will be used in the remaining of the paper. Specifically, Section 2.1 introduces the adaptive systems, Section 2.2 describes goals and operational requirements, and finally, Section 2.3 introduces REST Middleware.

2.1 Adaptive systems.

An adaptive system is an open system able to adapt its behavior according to changes in the environment or in parts of the system itself. Hence the adaptability is the ability of a software system to adapt efficiently and rapidly to any changes in the context or in the requirements.

Typically the development of a software system is completed within the life cycle where before delivery, requirements engineers, software designers and developers realize the system components. In modern software systems it is always difficult to predict the needs of users, so a single optimal configuration of the system is difficult to model and design.

It may be necessary to vary requirements run-time and then design the components of the application and implementation, all the same, always on the basis of changes arising from the external environment and the context, of course. For example, moving along a sensor network determines changes in the external

environment that might trigger the execution of applications and software components not provided at design time.

2.2 Operational Requirements.

Goals are objectives the system is intended to achieve through the cooperation of agents in the envisioned software and its environment [6]. A requirement is a goal assigned to an agent in the software design [1]. While functional requirements specify the functionalities to be implemented, generally non-functional requirements can determine decisions on the architectural model: for example, if the system must ensure security it is good to use a proxy to access protected data, if the system must integrate existing components it is good that has a distributed objects architecture and so on. An Operational Requirement captures the conditions under which a system component may or must perform an operation to achieve a goal. The operational requirements eventually belong to the set of non-functional requirements and describe the behavior of the system. They can be described by formal or semi-formal languages (machines or process algebra) or declarative (logic). Definition of operational requirements is basic in defining and describing the system analyzed in terms of the behavior more than the functionality to be provided.

2.3 REST Middleware.

Nowadays many vertical M2M solutions have been designed independently for different applications, making the current M2M market very fragmented, which inevitably hinders a large-scale M2M deployment. To decrease the market fragmentation there have been many efforts from different standardization bodies to define horizontal service layers.

The European Telecommunications Standards Institute (ETSI) has defined with the SmartM2M standard a middleware which has a RESTful architecture [9]. On the other side, OneM2M, where are collaborating more than 200 standardization bodies and companies, is defining a RESTful middleware which will have a global validity [8].

The proposed solutions provide RESTful middlewares which separate the applications from communication domain. The middlewares are accessible via open interfaces and enable the development of services and applications independently of the underlying network. In addition they provide several service capabilities to enable machine registration, synchronous and asynchronous communication, resource discovery, access rights management, group broadcast, etc.

All the resources in the RESTful middlewares are organized in standardized resource trees and can be uniquely addressed by a Uniform Resource Identifier (URI). Their representations can be transferred and manipulated with verbs (i.e., retrieve, update, delete, and execute).

3 Model for adaptive applications composition

In this section we propose a formal model for runtime composition of software applications in a sensor network.

The model is a graph based structure, the purpose of using a graph structure is to determine which apps to execute and how and where they will be executed, depending on variation in the context and hence on data detected by the sensor. The main advantage of using a graph structure is the possibility to use rewriting or grammar graph techniques for extracting subgraphs satisfying a given requirement. Requirements to be satisfied are high level requirements (mainly Operational), that are hence modeled on the graph structure. The graph describes a snapshot of all the available software plugins mainly characterized by the data detected by a sensor in the network (Data), the functionality to be executed (Process) and by the state of the device where it will be executed (Technology). Each transition has a cost due to parameters involved in the context. Hence we can assume that each plugin is modeled as a triple of elements with a function cost associated.

Definition 1 (Resource Super Graph (RSG)).

A Resource Super Graph is a direct Acyclic Graph $G = \{\mathbf{N}, \mathbf{A}\}$, where nodes \mathbf{N} are resources - $\mathbf{N} = \mathbf{D} \cup \mathbf{P} \cup \mathbf{T}$ ($\mathbf{D} = \text{Data}$, $\mathbf{P} = \text{Process}$, $\mathbf{T} = \text{Technology}$) - and arcs $a \in \mathbf{A}$ are such that:

1. $\mathbf{A} \subseteq (\mathbf{D} \times \mathbf{P}) \cup (\mathbf{P} \times \mathbf{T})$, i.e. "arcs connect data with process, process with technology, technology with data";
2. $a \in \mathbf{D} \times \mathbf{P}$ defines the variable cost v_c
 $a \in \mathbf{P} \times \mathbf{T}$ defines the fixed cost f_c
 $a \in \mathbf{T} \times \mathbf{D}$ detection of a new data variation

Each node in the graph can be distinguished as: *Data*, *Process*, *Technology*. *Data* are those detected by a sensor network; *Process* is the operation performed on the data that can belong to one among the following: preprocessing, processing plugins, etc. *Technology* identifies the network type and the characteristics of the mobile devices. Arcs in the graph link pairs of nodes based on the value of a cost function. The cost computation depends on several parameters, that can be the type of node, the cost of the process and so on.

The cost function associated with the triple (Data, Process, Technology) depends on the features of the given plugin but also on the state of the devices. It is defined as $f_c(DPT)$ of the triple (DPT), and is given by the sum of the cost of the two arcs connecting respectively D with P and P with T, $f_c = v_c + f_x$. The contribution of the variable cost depends on the characteristics of the available network and of the devices on which it is calculated. Where $v_c = \text{device}_c + \text{network}_c$. The cost of the network includes information about the state of the network at the time of receipt of the request of a plugin execution, such as connection delay, network bit rate, packet size, etc. The cost of the device is given by the amount of available RAM on the middleware, due to the

number of active connections. and by the cost of forwarding information when for example a middleware is not available so the request must be forwarded to another one. So $device_c = middleware_c - forwarding_c$. On the other hand, the costs of mobile device is given by the amount of available RAM and mass memory, the level of the device battery, as well as the geographic location (used to choose which middleware register).

The contribution of fixed cost fx_c depends on the characteristics of the plugin, such as size in bytes, computational complexity and so on. It remains unchanged if the plugin runs on the mobile device or if runs on the middleware.

Moreover, given a pair of starting and destination nodes, there are multiple paths connecting them, hence we can extract more "sub-graphs" from a Super-Graph. A path in the graph, i.e., a subgraph, identifies a sequence of apps to be executed with specifications concerning the features of the technology –the kind of network or of device–, and the type of process to execute each of them – where and how the app is executed. We need an algorithm to extract a subgraph according to the optimum condition, for example for extracting the subgraph that minimizes a cost function according to parameters depending on the nodes's features.

A Resource SubGraph (RSubG) is the graph extracted by RSG by executing the *Data Process Technology* DPT algorithm to select the path in the RSG with minimum cost. Among all the possible subgraphs of RSG, hence all the possible sequences of plugins to be executed we need to find the best path, with minimum cost function in order to determine the best sequence of plugins as triggered by a set of data detected by sensors.

DPT Algorithm. Let us now define the proposed algorithm. The *Data Process Technology* DPT algorithm schedules, manages and monitors the data/technologies and processes execution on the devices. Suppose the hardware infrastructure of the sensor network is made up as described below. It is composed by motes, with a limited memory and computation capabilities. Physical motes are mapped onto logical ones, and have a virtual image at middleware level. The features of the middleware are those of a REST middleware whose functionalities can be extended through the implementation of adhoc plugins. Each plugin will encode functionalities that can be run-time loaded, depending on the specific requirement triggered by an event that occurred. Sensors detection is managed at middleware level, where subscribers have to register and where updated data can be sent. At master level a scheduler plugin has to check and manage variations in the context and in data perceived from the sensors to decide which plugin or sequence of plugins to activate. The master plugin manages a runtime composition of plugins able to perform functionalities depending on data retrieved by sensors, but at the same time satisfying high level requirements modeled by triggering of events or being in a given state.

Communication among plugin occurs through the middleware that forwards requests, data, responses among plugging and sensors according to low level protocols while interaction is scheduled and managed by the high level master application.

To extract the shortest paths of the graph, and then the sets of nodes or sequence of plugins, we define the DPT algorithm to extract the shortest path made up of terms (Data, Process, Technology).

```

Data: A Resource SuperGraph (RSG)
Result: Resource SubGraph (RSubG)
1  $D \leftarrow$  data nodes;
2  $P \leftarrow$  process nodes;
3  $T \leftarrow$  technology nodes;
4 foreach  $i = 1$  to  $\min(D\_length, P\_length, T\_length,)$  do
5   |  $ShortestPath(D, P)$ ;
6   |  $ShortestPath(P, T, cost_t)$ ;
7   | select next data node;
8   |  $ShortestPath(P, D, cost_d)$ ;
9   | Evaluate plugin sequence;
10 end

```

Algorithm 1: Algorithm Data Process Technology DPT

Step 1 computes the shortest path following a starting node of Data type, Step 2 computes the shortest path following node of type P, step 4 computes shortest path following node of type T. Each step has as a parameter the function cost computed till the previous node. Shortest path extraction follows Djikstra algorithm [7].

The hardware infrastructure of the network is composed by nodes, with a limited memory and computation capabilities. Physical nodes are mapped onto logical ones, and have a virtual image at middleware level. The features of the middleware are those of a REST middleware whose functionalities can be extended through the implementation of adhoc plugins. Each plugin will encode functionalities that can be run-time loaded, depending on the specific requirement triggered by an event that occurred. Sensors detection is managed at middleware level, where subscribers have to be registered and where updated data can be sent. At master level a scheduler plugin has to check and manage variations in the context and in data perceived from the sensors to decide which plugin or sequence of plugins to activate. The master plugin manages an runtime composition of plugins able to perform functionalities depending on data retrieved by sensors, but at the same time satisfying high level requirements modeled by triggering of events or being in a given state.

4 Model instantiation

In this section we instantiate the model defined in Section 3 on the use case scenario that follows.

It is a cold winter evening, the temperature in the house is low, the heating system is activated to reach soon a temperature that will ensure comfort and well-being to Bob and Mary that are going to come back to after a busy working day. The blinds close to avoid the dispersion of heat. As soon as they get into

the house the lights turn on. Mary goes into the kitchen and set about making dinner; she turns the oven on that will soon to bake tasty pork shank, in the laundry the washer and dryer are temporarily suspended to avoid overload. Bob comes into the living room where the lights turn on. He is very tired so decide to sprawl on the sof and enjoy some videos. So he prepares the projector for watching the video taken by of his GoPRO while skying the previous Sunday on mountain holiday. The video projection begins and the lights turn dim to create soft lights. Later, Mary later went – as every evening – to the basement to train on sports equipment while waiting for dinner to be ready. The daily news flow on the monitor of the tapis roulant on which Mary is training. Through headset she listens directives of the exercises to be carried out according to the training program as a result of the control of the calories consumed in the days and of the physical activity already performed. Mary wears her heart rate and distance walked monitors for physical activity. When the goal of training daily is going to be reached, in the bathroom the heating is switched on, the whirlpool is switched on to enable Mary to practice proper relaxation after physical activity. Mary goes into the bathroom and the lights turn on while the basement lights and sports equipment are turned off. Meanwhile, in the garden, video surveillance cameras found two suspicious individuals climbing on the first floor and forcing a window to enter the house, despite the presence of people in the house. The images sent to the nearby police station trigger the alarm that promptly active forces to stop the thieves intrusion. A spark caused by a failure of the electrical systems in the garage makes burst fire and soon the garage is filled with dense smoke. The high level of smoke triggers the fire alarm that immediately reaches the nearest fire department to active the necessary reliefs.

The Use case scenario is modeled in a Resource Super Graph with all the possible triples of Data Process and Technology nodes. The algorithm DPT "on-the-Fly" extracts triples of nodes and hence activates plugins execution depending on the function cost.

For example the first situation: *It is a cold winter evening, the temperature in the house is low, the heating system is activated to reach soon a temperature that will ensure comfort and well-being to Bob and Mary that are going to come back to after a busy working day. The blinds close to avoid the dispersion of heat.* We have different paths that can be followed to orchestrate plugins. Modeled data node is temperature variation, but can also be light variation and position variation. Besides for each data node there exists different process nodes: if data retrieved is the temperature variation, process may be that of turning the heating system on, but can also be that of closing the blinds for avoiding dispersion of heat. After that, technology can be wi-fi, and the application can run on the house middleware or on the smartphone. Considering light variation as data node the process node can be the turning light on but also the closing of binds. So technology may be the execution of the process on the mobile phone or on a different device, the choice between several alternatives depends on the function cost. Different values of retrieved data and of function cost evaluation would

determine different selections of path in the graph this means that the plugins to be executed and their orchestration is different depending on context behavior.

5 Conclusion and future work

In this paper we introduced a model for building "on-the-Fly" architecture of software systems based on data, processes and technology in context-aware environments.

The model is based on a graph structure to represent data, processing of context aware application and technological features and by an algorithm for extracting the sequence of applications to be executed.

We instantiated the model on a sensor network environment and validated the algorithm on a running example in a smart home use case scenario. We are currently working on performing wide and complex experiments to validate and test the model.

References

1. D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Learning operational requirements from goal models. In *Proceedings of ICSE '09*, pages 265–275. IEEE Computer Society, 2009.
2. Javier Cubo, Guadalupe Ortiz, Juan Boubeta-Puig, Howard Foster, and Winfried Lamersdorf. Adaptive services for the future internet. *J. UCS*, 20(8):1046–1048, 2014.
3. Dominique Guinard, Iulia Ion, and Simon Mayer. In search of an internet of things service architecture: Rest or ws-*? a developers perspective. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326–337. Springer, 2012.
4. Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.
5. Amira Ben Hamida, Fabio Kon, Gustavo Ansaldi Oliva, Carlos Eduardo Moreira Dos Santos, Jean-Pierre Lorré, Marco Autili, Guglielmo De Angelis, Apostolos Zarras, Nikolaos Georgantas, Valérie Issarny, et al. An integrated development and runtime environment for the future internet. In *The Future Internet*, pages 81–92. Springer, 2012.
6. E. Letier and A. van Lamsweerde. Deriving operational software specifications from system goals. In *Proc. of SIGSOFT '02/FSE-10*, pages 119–128. ACM, 2002.
7. S Skiena. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Reading, MA: Addison-Wesley, pages 225–227, 1990.
8. J. Swetina, Guang Lu, P. Jacobs, F. Ennesser, and JaeSeung Song. Toward a standardized common m2m service layer platform: Introduction to onem2m. *IEEE Wireless Communications*, 21(3):20–26, June 2014.
9. Elvis Vogli, Mahdi Ben Alaya, Thierry Monteil, Luigi Alfredo Grieco, and Khalil Drira. An efficient resource naming for enabling constrained devices in smartm2m architecture. In *IEEE International Conference on Industrial Technology (ICIT) 2015*, pages 1832–1837, March 2015.