Architecture for orchestration of M2M services

Gustavo Pires, Mário Antunes, Daniel Corujo, Diogo Gomes, João Paulo Barraca, Rui Aguiar

Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal
{gmpp, mario.antunes, dcorujo, dgomes, jpbarraca}@av.it.pt, ruilaa@det.ua.pt

***Abstract*—The past few years, miniaturization has allowed us to imbue computers into everyday devices. This in turn has enabled these devices to communicate with each other, and in doing so, allows us to collect a wealth of information, more accurately and with greater availability than ever before. This phenomenon is known as the Internet of Things. It allows smart environments to truly behave in an intelligent manner by using information collected from the devices mentioned above. However, it's necessary to model how the gathered data will influence the behavior of a smart environment. This open problem can be approached as a machine to machine (M2M) orchestration.**

In this paper we present a new architecture for M2M orchestration. This new architecture will be based around a platform that creates orchestration processes through a graphical interface. Through this interface a business process execution language (BPEL) service will be made and deployed on an enterprise service bus (ESB). Alongside this, we are also developing a collection of services that will be used for the purposes of implementing a smart environment.

***Index Terms*—Orchestration, Machine to Machine, Smart Environments**

I. Introduction

In the past few years, we have seen a massive increase in the number of smart devices. In fact, according to the ICT KTN [1], the number of these devices is expected to increase worldwide from 4.5 billion in 2011 to 50 billion by 2020. These devices are able to communicate with each other, sending relevant information about their environment and coordinating their actions. This trend is known as the Internet of Things (IoT).

These connected devices can be seen as an unused source of context information. In this work we will consider context as defined by Abowd and Dey [2],

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

According to the definition, context information is virtually any type of information as long it is related to some entity presented in the considered environment.

One computational area that will benefit from this new source of context information is the area of Smart Environments. Smart Environments have lacked the autonomy and adaptability necessary for truly intelligent actions. This was in part caused by the absence of reasoning regarding the user's actions and status, in spite of the entire environment being flooded by huge amounts of information. However, context information cannot be obtained just by gathering this information. It first must be processed and seen what relation each information source has with each other so that the context related to that information can be inferred.

One way to look at this problem is as an M2M orchestration issue. M2M is the networking of intelligent small devices that manage themselves and exchange information between them without human intervention. Orchestration is the automated coordination of a group of systems, organized by a central entity. As seen in Figure 1, a smart environment is made of a group of independent services. These services can be organized as an orchestration process.

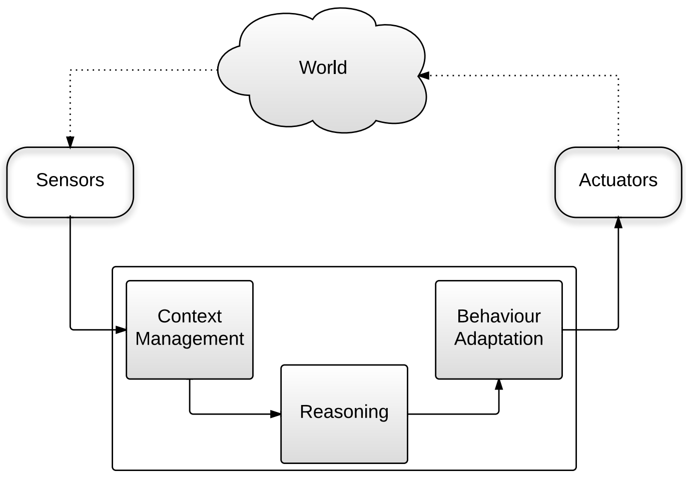

Fig. 1. Conceptual Architecture of a Smart Environment

In this paper we will present a novel architecture to enhance M2M orchestration. The proposed architecture is based around a platform that creates orchestration processes through a graphical interface. Through this interface it's possible to generate BPEL services by graphically connecting existing services. These services will then be deployed on an ESB and can also be used to compose new services. New services are also being developed for the purpose of supporting smart environments. Among others, services that extract information from a low level layer and services that reason about inhabitants' context.

A new reasoning services is also being developed to extract relevant knowledge from non-structured context information. By relevant knowledge, we mean the correct behavior of the environment. This knowledge will then be used by the smart environment's actuators to enforce those actions. The novelty of this service is the ability to use machine learning techniques as opposed to static ontologies or manually defined rules. This allows it to dynamically adapt to its present situation. For example, differences in the deployed environment would not mean that the system would need to be reconfigured, but instead it would dynamically adapt to the new environment.

The APOLLO project main objective is the development of a platform that supports new services in the area of machine-to-machine (M2M) communications. The project aims to develop a transversal technological platform that supports management, control and monitoring of an heterogeneous network of sensors and actuators. APOLLO exports a service layer to third parties willing to develop next generation M2M applications in various areas including Utilities, Transports, Health, Agriculture, Distribution and Consumer Electronics. The project platform will support from its start a vast set of M2M Smart Services & Applications such as Smart Metering, Smart Grids, m-Health (remote monitoring of patients), Smart Cities, Smart Home and Smart Buildings according to a Portuguese Government policy for the deployment of next generation networks.

In Section II we will discuss the current state of the art related to orchestration of M2M services and smart environments. Section III will expose the architecture, its advantages and drawbacks. In Section IV we will discuss the conceptual advantages of the proposed architecture and present future work related to this project.

II. State of the art

A Service Oriented Architecture (SOA) [3] supplies resources as self-contained, independent services that can then be reused. By doing so, these supplies can be maintained in a more strict fashion and be used by any number of other higher level services, even from an external entity to the one that created them.

Currently, in SOA implementations, orchestration has gained more popularity than choreography. The reason for this is that choreography presents several implementation problems, regarding communication overheads, even though it brings some advantages over orchestration [4]. In orchestration, only a central coordinator has to know the process, and the surrounding services can just worry about handling requests, not being involved in the management of process, and behave like any ordinary service. Choreography on the other hand, needs every process to know how it will intervene in the process, requiring information regarding other services' statuses. This means that choreography has potential to be more robust at the cost of added complexity.

The defacto standard language used to describe orchestration processes is BPEL [5]. This language describes each participating service's role and presents the work flow for the interactions between them. However, this language depends purely on services described through WSDL [1].

[1]http://www.w3.org/TR/wsdl

Barricelli et al. [6] proposed an architecture that allows domain experts to compose services through a graphical editor. The editor uses a semantic search engine to locate relevant services and an orchestration engine to handle the orchestration between these same services. The end product is a BPEL workflow document which can then be used to create services that execute the described process.

The typical scenario between participating services in a BPEL process is by definition an M2M scenario and so we can model any number of M2M scenarios as a BPEL process. As mentioned before, smart environments can be modeled as an M2M orchestration. The focus of this paper is on M2M orchestration of smart environments. Currently smart environments are developed as tightly coupled services that can only be orchestrated in a single manner. Below we present the most relevant smart environment projects.

Mozer [7], [8] applied neural networks [9] in home automation. The author developed a system that was able to control air, heating, lighting, ventilation and water heating. The main goal of the project was to anticipate inhabitants needs and conserve energy at the same time. For that, it applied reinforcement learning [9]. During a learning period, inhabitants indicated their preferences whenever predictions were incorrect by selecting themselves what they would expect the system to do (for instance, if they want the lights on, and the system did not turn them, users simply turned them on). This way, the system would adjust itself to its inhabitants preferences.

Vainio et al [10] proposed home-control system that uses fuzzy logic rules [11]. Initial rules were given manually and, through reinforcement learning, the home adapted by replacing old rules with new ones. This method was applied to control a lighting system in a smart-home laboratory environment. The author concluded that users did not care if the rules generated by the system didn't match exactly what they wanted. It also concluded that, after a certain amount of time, if a rule had a big weight, it was likely to keep its importance, and new rules, that correspond to sporadic actions, were quickly eliminated.

These smart environment systems use tightly coupled components. We hope that our proposal sparks interest in loosely coupled smart environment systems, and in doing so make it easier to build and customize these kinds of systems. Loosely coupled smart environments systems would also make it easier to add/remove/modify components during runtime.

III. Conceptual architecture

The proposed architecture can easily, through a graphic interface, orchestrate smart environments related services. These services can be split into two categories: reasoning services and context awareness services.

Reasoning services are services that process context information and imbue the environment with the necessary behavior to improve the inhabitants' quality of life. **Context awareness services** provide a bridge between the environment and the system. Through the sensors it can perceive the environment's context information, while at the same time it can use actuators to enforce changes onto the environment.

This section will be divided into three subsections relating to each of these kinds of services and to the orchestration application behind them. In Subsection III-A we integrate the necessary concepts for a flexible M2M communication framework. In Subsection III-B we present the reasoning mechanisms that will be used by the smart environment. Finally in Subsection III-C we show the orchestration application that will be able to use both of these in order to actually make the smart environment work.

A. Context Awareness

In order to not only access the information provided by different kinds of devices in Smart Environments, but to also allow their control, a flexible communication and interfacing framework needs to be set in place. Considering as well that such environments are of a widely heterogeneous nature, where involved devices support disparate sets of resources and are accessed through a myriad of networking technologies, the underlying communication framework is furthermore placed under stringent requirements: not only must it be able to simultaneously cater to the processing restrictions of devices with greater or lesser CPU and memory capabilities, but it has also to have the ability to provide information with a meaningful degree of usefulness despite those capabilities.

In this sense, our framework exploits the information gather and control capabilities of MINDiT [12], a flexible cross-layer interfacing architecture. This architecture leverages Media Independent concepts, such as the ones presented in IEEE 802.21 [13] and IEEE P1905 [14], where an abstraction layer allows the interexchange of information between lower layers (e.g., link access interfaces) and higher layers (e.g., application and service processes) under a common interface. In this sense, the conception and deployment of applications (and associated operating system mechanisms) aiming to obtain information and operate the different links, is simplified, with the middleware layer abstracting commands, events and information definition, not only locally but also remotely towards other agents located elsewhere on the network.

However, such media independent concepts are tailored concerning a well defined scope, where the commands, events and information elements supported by IEEE 802.21 concern handover optimization and, for the IEEE P1905 case, hybrid home networking. Achieving a flexible approach, MINDiT goes beyond the static nature of the interfacing capabilities of the previously mentioned technologies, and provides the means for dynamic interface definition and retrieval in true heterogeneous Smart Environments. Fig. 2 illustrates the deployment of the generic interfacing mechanisms provided by MINDiT on our framework, based on the concept proposed in [12].

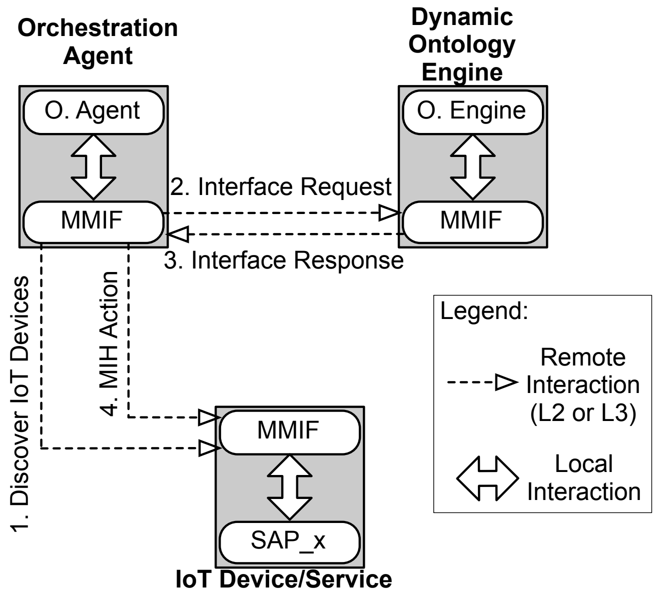

Fig. 2. MINDiT Interfacing Mechanism

The interfacing architecture of our framework relies on information exchanges between three different entities, composing an IoT Device or Service, an Orchestration Agent and an Ontology Engine.

- **IoT Device or Service:** The flexibility of our framework allows to reach for information, and execute control primitives, not only over physical aspects of a device (e.g., activate a mechanical arm to execute a precise movement, command a wireless interface to scan for access points), but also in services running in those devices or in network elements as well (e.g., switch video codec to reduce required bandwidth, trigger mobility handover procedures). The supported control and information gathered procedures are defined in the Service Access Point (SAP_x), composing the description of the list of supported commands by that device or service. To access a SAP, the node which composes the device or contains the service, is coupled with a MINDiT Media Independent Function (MMIF). This entity is a core component of this information interexchange architecture, aggregating a series of interfacing abstraction and control mechanisms:
    1) *Local and remote information exchange:* the MMIF acts as an intermediate abstraction control point for IoT communications, by receiving commands sent from agents (internal or external to the device), and collecting informational events from the SAPs. In this sense, it manages transaction state regarding these interactions, and is able to use Layer 2 and Layer 3 transport. Remote interaction is always executed between the MMIFs of the two separate entities involved.
    2) *Agent discovery and capability exchange:* the MMIF, when requested by a local or external agent,

is able to respond with the supported capabilities of the local node, regarding existing SAPs and respective supported commands. Likewise, it is also able to proactively broadcast this information in a beacon-like nature, allowing its discovery.

3) *Event registration:* agents are able to register for the reception of information events, when these are generated by the SAPs and conveyed to their local MMIF. Therein, the MMIF analyses if any other entity has requested for their reception and sends that information accordingly.

- **Orchestration Agent:** This agent composes a service or application entity implementing behaviour able to consume information and control from other devices or services. Through the knowledge of existence of other MINDiT-enabled entities (both remote and local), coupled with the knowledge of their supported commands and events provided by their SAP, the agents are able to execute the operations associated to their service logic, with support form MINDiT. It is important to notice that, in order to use these mechanisms, the agents need to be coupled with an MMIF to manage this information exchange. Moreover, the MMIF has to be seen as a complement component contained by the agent, and not as the core component itself (i.e., the agent uses MINDiT and not the other way around). In Fig. 2, the Orchestration Agent's logic establishing the bridge between its service logic and the operation with the MINDiT framework is pictured as a MINDiT Media Independent User (MMI-User).
- **Dynamic Ontology Engine:** The Ontology Engine composes the interaction with the orchestration and context structuring components of our framework, in terms of device and service interfacing. Concretely, the different capabilities of devices and services in IoT environments not only generate a plethora of different SAPs with different capabilities, but are also subject to different perceptions and interpretations of the raw data elements and parameters with which they interact. As an example, a temperature sensor can provide information in Celsius or in Fahrenheit. An Orchestration Agent needs to be aware that the numerical input sent by these devices can vary widely in definition. Moreover, the controlling capabilities also share this concern. As such, MINDiT utilizes the knowledge extracted from non structured information capabilities of our framework, to bridge the commands and parameters advertised by the SAPs of services and devices, to the necessities identified by the intended agent's behaviour.

The interactions involving these three entities are also identified in Fig. 2, following the following process:

1) **Discover IoT Devices:** Agents, motivated by their service behaviour, wish to interact with different IoT devices and/or services. Under our framework, the agents use MINDiT mechanisms to discover other MINDiT-enabled IoT entities. This discovery provides initial information indications that allow the agent to analyse which IoT entities fall within its scope of interest. For example, the announcement of the tags *#sensor* and *#temperature* provide insight to the agent that this entity deals with temperature sensing.
2) **Interface Request:** With this information, the agent is able to query the Dynamic Ontology Engine providing the received tags, alongside any meaningful information for the task belonging to its service logic (i.e., requirements, network domain, identifiers, used access technology, amongst others).
3) **Interface Response:** The Dynamic Ontology Engine extracts the necessary SAP interface from the available non structured information repository, which has been previously populated by service provides and device manufacturers with the necessary MINDiT SAP information. In this way, the MMI_User is able to gain knowledge of the SAPs for interfacing with the intended devices.
4) **MIH Action:** Empowered with the knowledge of the commands and parameters supported by the target IoT device or service, the agent is able to send a Media Independent Handover (MIH) Action command. The structure of this message is based on the one defined by the IEEE 802.21 standard regarding MIH commands. Unlike that standard, where each command has a separate structure defined by Type, Length and Value (TLV), in MINDiT we have leveraged a generic TLV structure able to identify the different commands made available by the different SAPs, as seen in Fig. 3. In this way, not only is the agent able to retain the MIH management mechanisms (e.g., transaction management) via the MIH Header as well as through indicating the MMIF Source and Destination identifiers, it is also able to specify the identifier for the target SAP, which is the intended Action and the resulting value. This ensures a more flexible approach in terms of defining the intended behaviour to be executed in the target IoT Device or Service, as verified in [12].

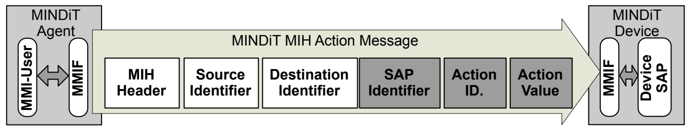

Fig. 3. MINDiT Generic MIHF Action Message

Through these mechanisms, our framework uses the MINDiT architecture to achieve the following capabilities:

1) To discover interfaceable IoT devices and services, along with their media independent link access connectivity information (e.g., IP address, MAC address, network domain, transport layer port, protocol, etc.);
2) To register agents into those IoT devices and services, allowing them to exchange connection establishment

data (e.g., to support authentication and other security processes) as well as to manifest interest in collecting informational events when triggered;

3) To dynamically request the interfacing primitives and parameters supported by the IoT device or service's SAP, by interacting with the non structured information repository from our framework;
4) To send those commands, and collect back information or command results, towards the target IoT device or service.

All this behaviour is achieved through a single generic message exchange definition, which can occur at either Layer 2 or Layer 3, according to the network and devices capabilities, which places a low impact on application service design, but providing still meaningful information content and capabilities to high-level processing. Contrary to other solutions employing resource-consuming service-oriented communication procedures that place stringent requirements over IoT devices (e.g., Devices Profile for Web Services (DPWS), Efficient XML Interchange (EXI) or Web Service Definition Language (WSDL), or on the other end, solutions that focus on wireless sensor link optimization aspects [15], MINDiT empowers our framework with a flexible, generic single protocol for IoT device interfacing.

B. Intelligent Systems

The APOLLO project's main objective is to export a service layer that allows third parties to develop next generation M2M applications. Smart environments are important applications that can be developed using the APOLLO service layer. In order to develop smart environments it is necessary to provide reasoning services. These reasoning services are necessary to comprehend the environment context through the sensors. Due to the heterogeneous nature of the information provided by the sensors and the dynamic ambient associated with smart environments it is difficult to develop efficient reasoning services. In this subsection we will discuss the difficulties of developing reasoning systems for smart environments and proposed a method that is able to cope with heterogeneous environments.

As previously mention, context information is any information that can be used to characterize an entity's situation. It is important to notice that this definition of context information does not specify any structure to share context information. This is strongly related with the heterogeneity associated with context information. Due to this heterogeneity, it is difficult to store context information into a relational database, as show in Figure 4. Context is better modeled as a continuous stream of information [16]. This approach has the advantage of facilitating the storage of the context information. On the other hand this approach does not enforce a relational model. The majority of knowledge extraction techniques rely on the relational model and the relations defined by it.

The most common approach to minimize this problem is to define an ontology. In computer science, an ontology is a description of the concepts and relations that can exist

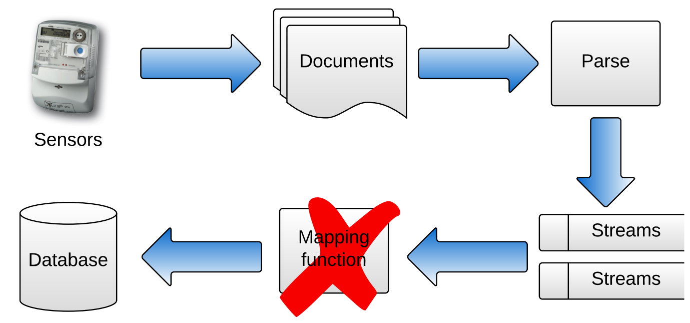

Fig. 4. Due to the heterogeneity associated to smart environments there is no dynamic function to correctly map an unstructured group of streams into a relational database.

between them [17]. Currently several context-aware systems use ontologies to map the continuous streams of data into a relation model. After that, conventional techniques can be applied in the model in order to extract meaningful knowledge. However, the use of ontologies implies a tradeoff between the quantity of represented concepts and the number of different scenarios the system is able to support. Another important issue about the use of ontologies is that users have to define all the relations between concepts. In other words the reasoning process becomes limited to the relations previously defined by users.

Currently we are developing a method to extract knowledge from unstructured context information that does not require static ontologies. The main objective of the proposed method is to learn the underlying model of unstructured information through machine learning techniques. The underlying model can be perceived as a dynamic ontology that is defined by the system itself.

Figure 5 shows the conceptual architecture of the platform. At this stage of development i is not relevant for the platform how the sensors send information. Document, in this context, represents a semi structured file (i.e. xml, json or cvs files) that contains data sent from some sensors. The platform analyses data from sensors and computes the underlying model, i.e. a dynamic ontology. The data is rearranged according to the underlying model. After this step conventional knowledge extraction methods are used to find relevant patterns. The majority of knowledge extraction frameworks provide support and confidence measures. These metrics are used by the model extraction process to determine the quality of the underlying model.

Figure 6 shows an expansion of the platform. The received documents are parsed and stored into a group of streams of data. Each stream is identified by the names extracted from the entity contained in the documents. E.g. if a document contains an entity named temperature and an associated value of 25, this value will be stored in a stream named temperature.

Two different analysis methods are used on the streams: statistic analysis and semantic analysis. First correlations matrices (statistic analysis) are computed based on the values on the streams. A strong mathematical correlation can be

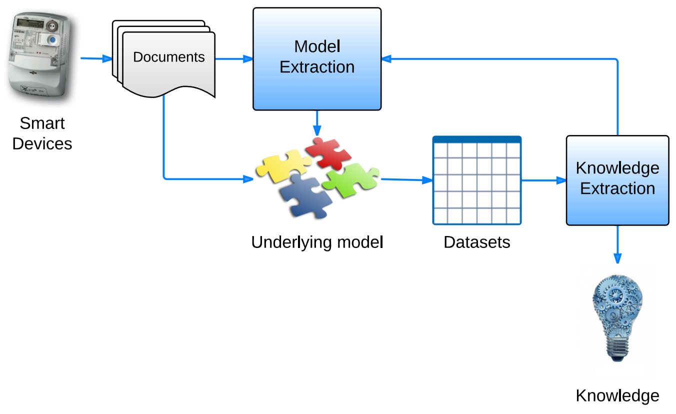

Fig. 5. Architecture of the knowledge extraction platform.

perceived as a relation on the underlying model.

After semantic analysis is used to discover relations between streams' names. The meaning of each streams' name is retrieved from a semantic web service. Text mining techniques are used to detect relations in the meanings of the streams' names. A graph of concepts is built based on the relations discovered. Each node in the graph represents a stream's name, while each edge represents a relations between two concepts.

After clustering algorithms are used to group the related streams. The results of the statistic (correlations matrices) and semantic analysis (graph of concepts) are the distance metrics used by the clustering algorithms. The underlying model is composed by the groups of concepts and relations returned by the clustering algorithm.

These type of analysis are subject to ambiguities in the concepts and relations. Association rules are used to detect relevant patterns in the rearranged data. The model extraction process uses the support and confidence framework to solve ambiguities in the concepts and relations. This feedback loop can be perceived as reinforcement learning.

Conceptually this approach offers some interesting advantages:

1) The proposed platform does not require the definition of static ontologies.
2) The dynamic model can cope with the evolution of concepts and relations.
3) The platform has the potential to discover unknown relations.

Currently we are finishing some details in the architecture of the knowledge extraction service. There are some open issues, e.g. how the platform should optimize based on the knowledge extraction feedback, what clustering algorithm should be used, what semantic services offers relevant information. These concerns are being addressed in a PhD Thesis. After solving these open issues the platform will be implemented as part of APOLLO project.

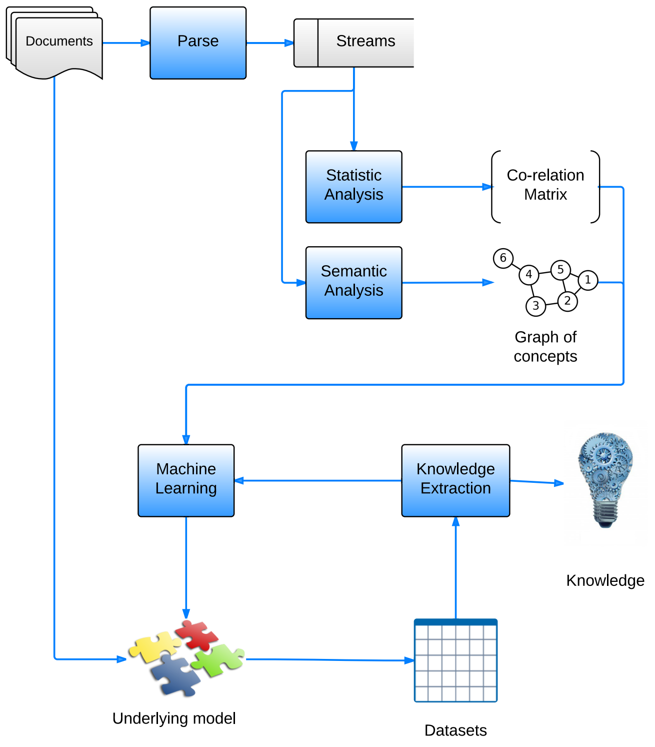

Fig. 6. Architecture of the model extraction process.

C. Orchestration

Smart environments need to cope with a diversity of underlying hardware and different user needs. One way to cope with this diversity is through the use of a SOA. A SOA allows components to be modules as independent services, creating a loosely-coupled system. It also allows the use of external modules that would be otherwise unavailable, such as proprietary services in which the implementer is not interested in revealing the internal logic. While this last one is beyond the scope of a typical smart environment, when one considers a city-wide deployment, having such capabilities starts having interesting applications.

One problem users face in a smart environment is that they currently might not be able to configure the behaviors they want the system to take. Since they may not be able to program, either time constraints or skills, the use of services together with a way of providing orchestration capabilities for them is one of the most promising options. As such, we decided to allow users with little programming skill, as well as domain experts to orchestrate services as they please through a supposedly easy to use graphical interface.

The orchestration architecture can be seen in Figure 7. We can divide it into four major components:

- Orchestration Creator Service
- Graphical Orchestration Interface
- Enterprise Service Bus
- Services

The first two are this paper's contribution to this orches-

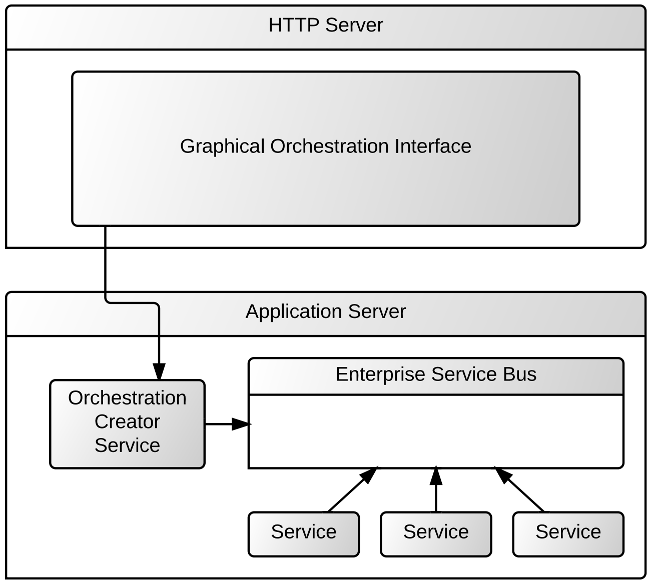

Fig. 7. The orchestration architecture

tration architecture. In fairly simple terms, the Graphical Orchestration Interface will obtain a list of available services. It will then provide a graphical representation of these services for the user to compose higher level services with. Afterwards, it creates a process description from the users input and call the Orchestration Creator Service. This service will then create and deploy a new service based on the obtained process description.

The Graphical Orchestration Interface is being implemented in HTML/JavaScript using Three Node JS[2], thus ensuring that most devices will be able to use it. The process description language to be used is most likely going to be BPEL. However, the current BPEL specification can only handle services described through WSDL. This limitation restricts the ability to orchestrate services that do not naturally use such description language, such as those using REST. These kinds of services are often much simpler and can be implemented using a wider variety of devices, increasing the scope of the Internet of Things. A number of BPEL extensions have been proposed over the years to address this restriction [18]–[22]. Regarding this issue, we are still in the process of researching how we should make BPEL processes interact with WSDLless services such as REST.

As for the ESB, we'll use is JBoss's Switchyard[3] due to a business requirement and as such the we'll also use JBoss AS[4].

IV. Conclusions

Current smart environments are not flexible enough to be considered truly intelligent. They use tightly coupled components and cannot cope with changes in the smart environment, e.g. adding new sensors to the system. In this paper we introduced a new conceptual architecture for M2M orchestration that conceptually solves these issues. With it, we proposed a new intelligent system to extract context from non-structured information and a new orchestration mechanism to allow users of little programming skill to create processes based on deployed services.

We hope this architecture will facilitate the proliferation loosely coupled smart environments that do not rely on static ontologies or only user defined rules. Our orchestration platform enables the use of a set of services that ranges from user defined rules to truly intelligent systems. Additionally, the move to a SOA will help standardize the way smart environment are controlled and allow the collaboration of several smart environments, potentially enabling smart environments of arbitrary sizes to be built with ease.

Currently, we're finishing an implementation of the context management component based on the XMPP protocol and key-value databases. Afterwards, we'll design the reasoning services provided by the APOLLO platform. As previously mentioned, we are still trying to make BPEL processes interact with WSDLless services such as REST. After analysing Switchyard's framework, we decided to implement a new component that enable services to access the service registry.

[2]https://github.com/idflood/ThreeNodes.js
[3]http://www.jboss.org/switchyard.html
[4]http://www.jboss.org/jbossas

V. Acknowledgement

This work has been partially funded by the Portuguese Innovation Agency/ National Strategic Reference Framework (AdI/QREN) under grant agreement No. 2011/021580 (APOLLO project).

References

[1] U. F. I. S. Group, "Future internet report," ICT Knowledge Transfer Network, Tech. Rep., May 2011.
[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, ser. HUC '99. London, UK,: Springer-Verlag, 1999, pp. 304–307. [Online]. Available: http://dl.acm.org/citation.cfm?id=647985.743843
[3] M. Bell, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley Publishing, 2008.
[4] S. Quinton, I. Ben-Hafaiedh, and S. Graf, "From orchestration to choreography: Memoryless and distributed orchestrators," in *FLACOS'09 Workshop Proceedings*, 2009.
[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Bpel4ws, business process execution language for web services," IBM, Tech. Rep., 2003.
[6] B. R. Barricelli, P. Mussio, S. Valtolina, M. Padula, P. L. Scala, and A. Piccinno, "Visual workflow composition through semantic orchestration of web services," in *Proceedings of the International Conference on Advanced Visual Interfaces*, ser. AVI '10. New York, NY, USA: ACM, 2010, pp. 405–405. [Online]. Available: http://doi.acm.org/10.1145/1842993.1843079
[7] M. Mozer, "The neural network house: An environment that adapts to its inhabitants," in *Proceedings of the American Association for Artificial Intelligence*, A. Press, Ed., 1998, pp. 110–114.
[8] ——, "Lessons from an adaptive home," in *Smart Environments: Technology, Protocols and Applications*, D. J. Cook and S. K. Das, Eds. John Wiley & Sons, Inc., 2005, pp. 271–294.
[9] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[10] A.-M. Vainio, M. Valtonen, and J. Vanhala, “Proactive fuzzy control and adaptation methods for smart homes,” *IEEE Intelligent Systems*, vol. 23, pp. 42–49, 2008.

[11] J. Mendel, “Fuzzy logic systems for engineering: a tutorial,” *Proceedings of the IEEE*, vol. 83, no. 3, pp. 345 –377, mar 1995.

[12] D. Corujo, M. Lebre, D. Gomes, and R. Aguiar, “Mindit: A framework for media independent access to things,” *Computer Communications*, vol. 35, no. 15, pp. 1772 – 1785, 2012, smart and Interactive Ubiquitous Multimedia Services. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366412000734

[13] IEEE, “Ieee standard for local and metropolitan area networks- part 21: Media independent handover,” *IEEE Std 802.21-2008*, 21 2009.

[14] “Ieee draft standard for a convergent digital home network for heterogeneous technologies,” *IEEE P1905.1/D06, September 2012*, pp. 1 –125, 11 2012.

[15] W. Masri and Z. Mammeri, “Middleware for wireless sensor networks: A comparative analysis,” in *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, sept. 2007, pp. 349 –356.

[16] N. Santos, . O. M. Pereira, and D. Gomes, “Context storage using nosql,” in *Proc. 2011 Conferência sobre Redes de Computadores*, Coimbra, Portugal, November 2011.

[17] T. Gruber, “Toward principles for the design of ontologies used for knowledge sharing,” *International Journal Human-Computer Studies*, vol. 43, no. 5-6, pp. 907–928, November 1995.

[18] C. Pautasso, “Bpel for rest,” in *Proc. of the 6th International Conference on Business Process Management (BPM 2008*, 2008.

[19] ——, “Restful web service composition with bpel for rest,” *Data Knowl. Eng.*, vol. 68, no. 9, pp. 851–866, September 2009.

[20] T. A. S. Foundation, “Restful bpel, part i,” 9 2012. [Online]. Available: http://ode.apache.org/restful-bpel-part-i.html

[21] ——, “Restful bpel, part ii,” 9 2012. [Online]. Available: http://ode.apache.org/restful-bpel-part-ii.html

[22] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann, “Bpel light,” in *5th International Conference on Business Process Management (BPM 2007)*. Springer, Sep. 2007.