RESTful Integration of Heterogeneous Devices in Pervasive Environments

Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, Frank Eliassen

RESTful Integration of Heterogeneous Devices in Pervasive Environments

Daniel Romero[1], Gabriel Hermosillo[1], Amirhosein Taherkordi[2], Russel Nzekwa[1], Romain Rouvoy[1], and Frank Eliassen[2]

[1] ADAM Project-Team
University of Lille 1, LIFL
INRIA Lille – Nord Europe
F-59650 Villeneuve d'Ascq
firstname.lastname@inria.fr

[2] Department of Informatics
University of Oslo
N-0316 Oslo
amirhost@ifi.uio.no, frank@ifi.uio.no

Abstract. More and more home devices are equipped with advanced computational capabilities to improve the user satisfaction (*e.g.*, programmable heating system, Internet TV). Although these devices exhibit communication capabilities, their integration into a larger home monitoring system remains a challenging task, partly due to the strong heterogeneity of technologies and protocols. In this paper, we therefore propose to reconsider the architecture of home monitoring systems by focusing on data and events that are produced and triggered by home devices. In particular, our middleware platform, named DIGIHOME, applies *i)* the REST (*REpresentational State Transfer*) architectural style to leverage on the integration of multi-scale systems-of-systems (from Wireless Sensor Networks to the Internet) and *ii)* a CEP (*Complex Event Processing*) engine to collect information from heterogeneous sources and detect application-specific situations. The benefits of the DIGIHOME platform are demonstrated on smart home scenarios covering home automation, emergency detection, and energy saving situations.

1 Introduction

Pervasive environments support context-aware applications that adapt their behavior by reasoning dynamically over the user and the surrounding information. This contextual information generally comes from diverse and heterogeneous sources, such as physical devices, *Wireless Sensors Networks* (WSNs), and smartphones. In order to exploit the information provided by these sources, an integration middleware is required to collect, process, and distribute the contextual information efficiently. However, the heterogeneity of systems in terms of technology capabilities and communication protocols, the mobility of the different interacting entities and the identification of adaptation situations makes this integration difficult. Thus, we need to provide a flexible solution in terms of communication and context processing to leverage context-aware applications on the integration of heterogeneous context providers.

In particular, a solution dealing with context information and control environments must be able to connect with a wide range of device types. However, the resource scarceness in WSNs and mobile devices makes the development of such a solution very challenging. In this paper, we propose the DIGIHOME platform, a simple but efficient middleware solution to facilitate context-awareness in pervasive environments. Specifically, DIGIHOME provides support for the *integration*, *processing* and *adaptation* of the context-aware applications. Our solution enables the integration of heterogeneous computational entities by relying on the REST (*REpresentational State Transfer*) principles [8], standard discovery and communication protocols, and resource representation formats. We believe that the REST concepts of simplicity (in terms of interaction protocols) and flexibility (regarding the supported representation formats) make it a suitable architecture style for integration in pervasive environments. Furthermore, while our solution also benefits from WSNs to operate simple event reasoning on the sensor nodes, we rely on *Complex Event Processing* [19] for analyzing in real-time the relationships between the different collected events and trigger rule-based adaptations.

The rest of this paper is organized as follows. We start by describing a smart home scenario in which we identify the key challenges in pervasive environments that motivate this work (cf. section 2). We continue by the description of DIGIHOME, our middleware platform to support the integration of systems-of-systems in pervasive environments (cf. section 3). Then, we discuss the benefits of our approach (cf. section 4) before discussing the related work (cf. section 5). Finally, we conclude by presenting some promising perspectives for this work (cf. section 6).

2 Background and Motivations

This section introduces the application and architectural foundations of our work in sections 2.1 and 2.3, respectively.

2.1 Motivating Scenario

A smart home generally refers to a house environment equipped with several types of computing entities, such as *sensors*, which collect physical information (temperature, movement detection, noise level, light, etc.), and *actuators*, which change the state of the environment. In this scenario, we consider a smart home equipped with occupancy, smoke detection, and temperature sensors. These tiny devices have the ability of collecting context information and communicating wirelessly with each other, in order to identify the context situation of the environment. In addition to that, we can also find actuators to physically control lights, TV, and air conditioning. Figure 1 illustrates the integration of these sensors and actuators in our scenario. As appreciated in this figure, the different entities use heterogenous protocols to interact. In the scenario, the smart phones provide information about user preferences for home configuration. Conflicts between the user preferences are resolved by giving priority to the person who arrived first to the room. The mobile devices also have an application that enables the control of the actuators present in the different rooms. This application can be adapted when there are changes in the actuator's configuration. Finally, there is a *Set-Top Box* (STB) which is able to gather information, and interact with the other co-located devices.

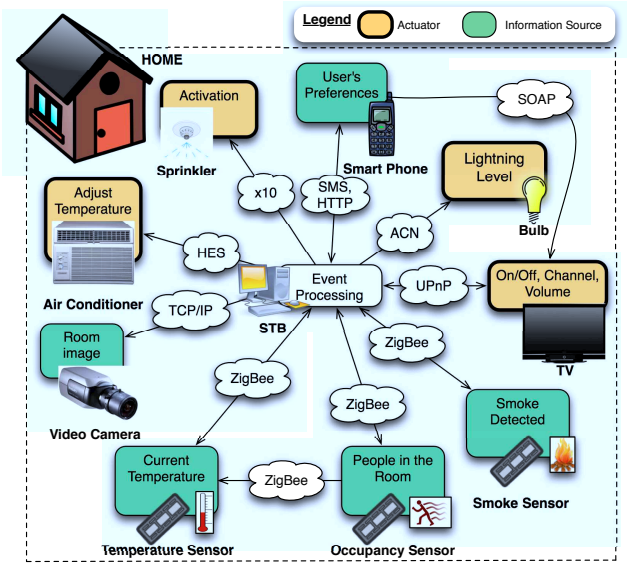

Fig. 1. Interactions between the smart home devices.

Situation 1: Alice arrives to the living room. The occupancy sensor detects her presence and triggers the temperature sensors to decrease the sampling rate of data. It also notifies the STB that the room is occupied by somebody, which in turn tries to identify the occupant by looking for a profile in her mobile device. When Alice's profile is found, the STB loads it and adjusts the temperature and lightening level of the room according to Alice's preferences.

Situation 2: The sensors detect smoke and notify the STB, which using the occupancy sensor, detects that the house is empty. The STB therefore notifies Alice via an SMS, and includes a picture of the room, captured using the surveillance camera. After checking the picture, Alice decides to remotely trigger the sprinklers using her mobile device. She also tells the system to alert the fire department about the problem. If Alice does not reply to the STB within 5 minutes, the system activates automatically the sprinklers and alerts the fire department.

Situation 3: Alice installs a new TV in the bedroom. The STB detects the presence of the new device, identifies it, and downloads the corresponding control software from an Internet repository. The platform tries to locate the available mobile devices and finds Alice's mobile device. The STB propose to update the mobile device with the components for controlling the new TV.

2.2 Key Challenges

The above described situations allow us to identify several key challenges in terms of:

1. *Integration of multi-scale entities*: The mobiles devices and sensors have different hardware and software capabilities, which make some devices more powerful than others. Therefore, this heterogeneity requires a flexible and simple solution that supports multiple interaction mechanisms and considers the restricted capabilities of some devices. In particular, regarding sensor nodes, the immaturity of high-level communication protocols, as well as the inherent resource scarceness, bring two critical challenges to our work: *1)* how to connect sensor nodes to mobile devices and actuators through a standard high-level communication protocol, and *2)* the framework which runs over sensor nodes for supporting context-awareness and adaptation should not impose high resource demands.
2. *Entities mobility*: In our scenario, computational entities appear and disappear constantly. In particular, mobile devices providing user profiles are not always accessible (they can be turned off or the owner can leave the house with them). In a similar way, the actuators can be replaced or new ones can be added. Thus, we need to provide the functionality to discover new entities dynamically and to support device disconnections.
3. *Information processing and adaptation*: In order to support adaptation, we first need to identify the situations in which the adaptation is required. We have a lot of information that is generated by the different devices in the environment and we need to define which part of this information is useful in order to identify relevant situations and react accordingly. In our scenario, those situations include the load of Alice's profile and the adjustment of the temperature, the sending of an alert via SMS in case of an emergency, and the adaptation of Alice's mobile device to control the new TV in her bedroom.

2.3 REST: REpresentational State Transfer

REpresentational State Transfer (REST) is a resource-oriented software architecture style identified by R. Fielding for building Internet-scale distributed applications [8]. Typically, the REST triangle defines the principles for encoding (*content types*), addressing (*nouns*), and accessing (*verbs*) a collection of *resources* using Internet standards. Resources, which are central to REST, are *uniquely addressable* using a universal syntax (*e.g.*, a URL in HTTP) and share a *uniform interface* for the transfer of application states between client and server (*e.g.*, `GET/POST/PUT/DELETE` in HTTP). REST resources may typically exhibit multiple typed representations using—for example—XML, JSON, YAML, or plain text documents. Thus, RESTful systems are loosely-coupled systems which follow these principles to exchange application states as resource representations. This kind of stateless interactions improves the resources consumption and the scalability of the system.

According to R. Fielding [8], *"REST's client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST*

enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability."

Synthesis. REST identifies an efficient architectural style for disseminating resources, which can be encoded under various representations. Therefore, we believe that REST provides a suitable framework for mediating and processing context information in an efficient and scalable manner.

3 The DIGIHOME Platform

In order to address the challenges introduced in section 2.2, we propose the DIGIHOME middleware platform for pervasive environments. The main objective of DIGIHOME is to provide a comprehensive and simple solution for dealing with context processing in this kind of environments. To do that, the platform offers services for *integration* of context information and the detection of *adaptation situations* based on this information. DIGIHOME also benefits from the WSNs capabilities to process simple events and make local decisions when possible. With DIGIHOME, we can support variants of the platform for resource-constrained devices (sensor nodes or mobile devices in our scenario) that interact with powerful variants running on more powerful devices (*e.g.*, the STB in the smart scenario).

Figure 2 depicts the general architecture of DIGIHOME. In this architecture, the Event Collector retrieves and stores the recent information produced by context collectors, such as mobile devices or sensors. The CEP Engine is responsible for event processing and uses the Decision Executor to perform actions specified by the Adaptation Rules. In DIGIHOME, the integration of the heterogenous entities is achieved via the *RESTful Communication* middleware framework that provides software connectors following the REST principles. In the rest of this section, we give more details about the integration of the information via REST (cf. section 3.1), the complex event processing (cf. section 3.2) and the distribution of DIGIHOME platforms in WSNs (cf. section 3.3). Finally, section 3.4 reports optimizations that are applied to the platform in order to control the reactivity and the stability of the system.

3.1 RESTful Communication Middleware

The integration challenge identified in section 2.2 requires a flexible infrastructure enabling communication and discovery between all the participants (*i.e.*, mobile devices, sensor nodes, actuators, and the set-top box). To address this issue, we classify the heterogeneity in terms of resources and different interaction mechanisms. This communication middleware that we define in DIGIHOME therefore follows the REST principles. The simplicity, lightness, reusability, extensibility, and flexibility properties that characterize REST make it a suitable option for context dissemination in pervasive environments.

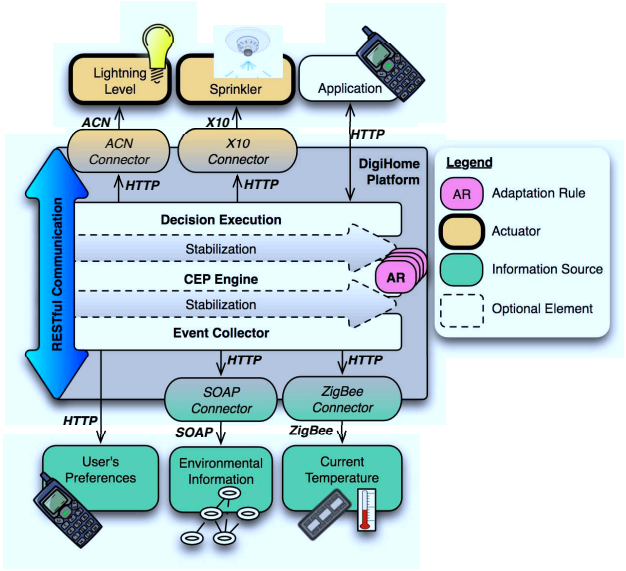

Fig. 2. Description of the DIGIHOME architecture.

The DIGIHOME communication middleware defines ubiquitous connectors encapsulating the distribution concerns. Software connectors [5,30] isolate interactions between components—*i.e.*, they support the transfer of control and data. The connectors can also provide non-functional services, such as persistency, messaging, and invocation helping to keep the component functionality focused on the domain specific concerns. In this way, the connectors foster the separation of concerns [30]. In the context of DIGIHOME, software connectors do not impact the event processing and support multiple implementations of the communication mechanism. Figure 2 depicts several examples of connectors for supporting protocols such as ZigBee [37], SOAP, and ACN [6]. For purposes of our scenario, we choose HTTP as the default interaction protocol for the middleware platform and the mobile devices. The choice of HTTP is motivated by its simplicity and the possibility to have mobile devices not only as consumer but also as service providers [21,25,31]. Our RESTful communication middleware supports additional protocols, such as XMPP [27] and Twitter [20] that can be more suitable in other situations.

The connectors also support spontaneous interoperability [16] to deal with the volatility of pervasive environments. As already mentioned in section 2, mobile devices, sensors, and actuators can continuously appear or disappear from the landscape. Therefore, the DIGIHOME connectors deal with this volatility by means of standard discovery protocols. By exploiting the extension capabilities of these discovery protocols, we can, for example, improve the context information advertisements with *Quality of Context* (QoC) [18] attributes for provider selection purposes. In particular, in our scenario we use UPnP [33] to discover mobile devices and actuators. We have selected this protocol

as several available electronics devices already support UPnP. Furthermore, although UPnP is an XML-based protocol, its application in WSNs does not impact the energy consumption because we do not need to process XML descriptions in sensor nodes (this is the responsibility of CEP Engine in DIGIHOME), just provide them.

3.2 Complex Event Processing

Complex Event Processing (CEP) is a technology for detecting relationships, in real-time, between series of simple and independent events from different sources, using predefined rules [35]. In our scenario, we consider a lot of heterogeneous devices (sensors, mobile devices, etc.) that generate isolated events, which can be used to obtain valuable information and to make decisions accordingly. We can see some examples of this in the scenario, like activating the sprinklers in case of detecting a fire or like updating the mobile device in order to control the new TV.

To manage those events, we need a decision-making engine that can process them and that can create relations to identify special situations, using predefined rules. In order to identify the desired events, the CEP Engine requires to communicate with an Event Collector, which is in charge of dealing with the subscriptions to the event sources. When an adaptation situation is detected, a corresponding action is triggered, which can go from an instruction to an actuator, to the adaptation of the system by adding or removing functionality. These actions are received by the Decision Executor, which has the responsibility of communicating with the different actuators in the environment.

In DIGIHOME, for the event processing in the set-top box, we use ESPER [7], a java open source stream event processing engine, to deal with the event management and decision making process. We chose ESPER for our platform because is the most supported open source project for CEP and is very stable, efficient, and fairly easy to use. The following code excerpt shows an example of an ESPER rule:

```
select sum(movement)
from MovementSensorEvent.win:time(60 sec)
```

This is a rule related to the scenario presented in section 2.1. Here we can see the use of a time window, which is a moving interval of time. The rule collects all the events from the movement sensor from the last 60 seconds.

3.3 Support for Wireless Sensor Networks

In DIGIHOME, there are two scopes for event processing: *local event processing*, and *global event processing*. To improve the efficiency of the system, the sensor nodes in our configuration can be considered as sensor networks in order to avoid assigning local decisions to the STB, responsible for global concerns. Specifically, the event generated by a sensor node may be of interest to a node within the sensor network. Therefore, instead of going through the centralized server framework for making decisions, the WSN itself takes the responsibility of processing events in a more efficient way. The architecture of DIGIHOME in sensor nodes supports both local event processing and

global event forwarding. The latter delegates the decision of global event processing to the STB. In the former case, the layered framework on the sensor node has the ability to connect directly to other sensor nodes in the environment and to deliver an event to the nodes subscribed to that type of event. Furthermore, the framework provides a lightweight mechanism for event processing in order to keep resource usage at a very low level. The execution layer also benefits from our unified communication protocol to send the configuration and adaptation instructions across the WSN. The event manager layer of our framework enables in-WSN decisions, whenever an event should be processed with other relevant events generated by other sensor nodes. As an example, when a temperature sensor senses a high degree, for deducing on fire detection, it needs to become aware of the smoke density in the room. Thus, the collaboration at network management layer becomes an essential need.

3.4 Platform Optimizations

Stabilization Algorithms. When system events are gathered from different sensors, they are forwarded to the CEP Engine, which analyses them before deciding which actions should the system perform. This decision-making task is often a costly procedure for the system and thus requires optimization techniques in order to optimize this task. One of these technics can consist in stabilizing the data flow, for example between the Event Collector and the CEP Engine. The role of the stabilization mechanism is therefore to filter events, preventing useless triggering of the decision-making task.

In [22], stabilization mechanisms are defined as algorithms and techniques that operate the system reconfigurations or adaptations only when relevant changes occur. In the smart home scenario (cf. section 2.1) the stabilization mechanism can be useful at several levels of our architecture. Typically, we can aggregate context information (*e.g.*, the user's preferences) or compute the average of some data (*e.g.*, the temperature). We also have the possibility of introducing stabilization mechanisms between the CEP Engine and the Decision Executor in order to avoid the recurrent triggering of unnecessary adaptations (cf. Figure 2).

Concerning the implementation of the stabilization mechanisms in our framework, we use the flexible approach proposed in [23]. This approach suggests a composition model, which consists of two modalities: *horizontal composition* and *vertical composition*. Horizontal composition consists in executing several stabilization algorithms concurrently, while the vertical composition refers to the sequential application of two or more algorithms on the same data sample.

Web Intermediaries. REST enables *Web Intermediaries* (WBI) to exploit the requests exchanged by the participants in the communication process. WBI are computational entities that are positioned between interacting entities on a network to tailor, customize, personalize, or enhance data as they flow along the stream [15]. Therefore, we can benefit from this opportunity to improve the performance of DIGIHOME. When the provided context information does not change much in time, the messages containing this information can be marked as cacheable within the communication protocol. This kind of annotation enables WBI caches to quickly analyze and intercept context requests always returning the same document. A similar optimization applies to security issues and the

filtering of context requests. Indeed, by using proxy servers as WBI, we can control the requested context resources and decide whether the incoming (or outgoing) context requests need to be propagated to the web server publishing the context resource. Other kinds of WBI can also be integrated in the system to operate, for example, resource transcoding, enrichment, or encryption.

4 Empirical Validation

Although the contribution of this paper lies in the adoption of a versatile architecture style for integrating the diversity of device appliances available in the pervasive environments, we have also made a performance evaluation of a prototype, implementing the proposed platform. This experimentation demonstrates the reasonable overhead imposed by the DIGIHOME platform.

4.1 Implementation Details

We built a prototype of the DIGIHOME platform based on the FRACTAL component model and we used the JULIA[3] implementation of the FRACTAL runtime environment [2]. In order to test our system, we measured the communication and discovery overheads using our RESTful approach as well as the event processing cost when using ESPER. To obtain these results, we have implemented the scene 1 of the smart home scenario (described in section 2.1).

4.2 Discovery and Communication Overhead

Table 1 reports the average latency for the context dissemination via REST. In this setup, we retrieve the user preferences from multiple providers (Nokia N800 Internet Tables and MacBook Pro) and use multiple formats for the context information (*i.e.*, XML, JSON, and Java Object Serialization). To do that, we have installed a lightweight version of the DIGIHOME platform that includes only the RESTful Communication Middleware as well as the Event Collector. We also measured the delay for discovering the information provided by the sources. For discovery, we selected the UPnP and SLP [11] protocols. In the tests, the platform aggregates the user's preferences to reduce the number of messages exchanged between the provider and the consumer. The measured time corresponds to exchange of REST messages as well as the marshalling/unmarshalling of the information. The cost of executing others protocols, such as ACN and ZigBee was not considered in this paper. The reader can find more information about the overhead introduced by these protocols in [1]. Furthermore, in this experimentation, the preferences retrieval from mobile devices via XML was not possible due to a limitation of the Java Virtual Machines used on the mobile device (CACAOVM & JamVM). Nevertheless, this is not a problem for our approach since several representations are available for the same preference.

[3] JULIA: `http://fractal.ow2.org/julia`

Client/ Provider(s)	Retrieval Latency			Notification Latency			Discovery Latency	
	Object (ms)	JSON (ms)	XML (ms)	Object (ms)	JSON (ms)	XML (ms)	SLP (ms)	UPnP (ms)
MacBook Pro (Local)	**74.3**	85.5	92.5	**78.6**	90.46	97.77	44.03	59.51
MacBook Pro/ MacBook Pro	**146.2**	158.3	165.3	**154.67**	167.48	174.88	63.62	120
MacBook Pro/ N800	**339.6**	375.75	N/A	**359.26**	397.54	N/A	128.99	136

Table 1. Performance of the RESTful connectors.

4.3 Event Processing Overhead

The time for context dissemination as well as for discovery confirms that DIGIHOME can integrate heterogeneous entities with a reasonable performance overhead. Furthermore, according to the documentation provided by ESPER[7], the efficiency of the engine to process the events is such that it exceeds over 500,000 events per second on a workstation and between 70,000 and 200,000 events per second running on an average laptop. The efficiency of the engine makes that the use of event processing in our system can be done at a low cost and given the modularity of our architecture, the ESPER engine can be installed in the device that provides the highest processing power. In the context of the DIGIHOME platform, we observed that ESPER took $1ms$ on average to process the adaptation rules.

5 Related Work

5.1 Context Dissemination

In literature, it is possible to find two kinds of solutions to deal with context integration: *centralized* and *decentralized*. In the centralized category we can find middleware, such as PACE [13] and *Context Distribution and Reasoning* (ConDoR) [26]. PACE proposes a centralized context management system based on repositories. The context-aware clients can interact with the repositories using protocols, such as Java RMI or HTTP. For its part, ConDoR takes advantage of the object-oriented model and ontology-based models to deal with context distribution and context reasoning, respectively. In ConDoR, the integration of the information using different protocols is not considered as an important issue. The problem with this kind of approach is the introduction of a single point of failure into the architecture, which limits its applicability to ubiquitous computing environments.

On the other hand, in decentralized approaches we can find solutions like CORTEX [29] and MUSIC [14,17]. CORTEX defines sentient objects as autonomous entities that have the capacity of retrieving, processing, and sharing context information using HTTP and SOAP. MUSIC middleware is another decentralized solution that proposes a peer-to-peer infrastructure dealing with context mediation. The decentralized approaches face the problem of fault tolerance by distributing the information across several machines. However, as well as some centralized solutions, the lack of flexibility in terms of the communication protocols remains a key limitation for these approaches.

In addition to that, peer-to-peer approaches have performance and security problems. In DIGIHOME, we provide a decentralized solution, where the different interacting devices can process the events retrieved from the environment. Furthermore, in DIGIHOME we provide flexibility in terms the interaction by supporting different kinds of communication protocols and we also allow spontaneous interoperability.

5.2 Complex Event Processing

Given the increasing interest to integrate the flow of data into the existing systems, CEP has gained some attention as it can help to provide that integration transforming isolated data into valuable information. In this context we can find some works similar to ours in [3] and [36]. In [3], the authors integrate CEP into their existing project called SAPHE (*Smart and Aware Pervasive Healthcare*), and also use ESPER as their CEP engine. As the project name shows, the project is applied to healthcare and use sensors to monitor a patient's activity and vital signs. They use CEP to correlate and analyze the sensor data in order to calculate critical factors of the patient locally in their set-top box, without having to send all the events to an external server. In their approach they lack a way to discover new services and they never mention how, if possible, would they interact with actuators in order to adapt to the context and respond to a specific situation.

An *Event-Driven Architecture* (EDA) that combines the advantages of WSN with CEP is presented in [36]. They use an extension of the RFID EPCglobal architecture which allows the interaction of RFID and WSN events. Once the events are collected, they use CEP to detect specific situations. They use a smart shelf application as their scenario to show how the events from both sources can be combined. Even though both technologies seem to interact in their project, their specification is somehow limited because they do not specify how the information obtained could be used, other than generating a report that will be logged in the EPCIS server.

5.3 Wireless Sensor Networks

In [32], the authors describe a WSN-specialized resource discovery protocol, called DRD. In this approach, each node sends a binary XML description to another node that has the role of *Cluster Head* (CH). The CH is selected among all the nodes based on their remaining energy. Therefore, it is necessary to give all the nodes the capacity of being a CH. Consequently, all nodes need an SQLlite database, libxml2 and a binary XML parser in order to implement the CH functionalities. In DIGIHOME, with our modular architecture, we consider the resource constraint of sensors nodes and provide a lightweight version of the platform that delegates complex processing to more powerful devices. Therefore, not all the nodes have to be CH. Furthermore we benefit from the advertisement capacities of sensor nodes to identify adaptation situations.

In CoBIs [4], business applications are able to access functionalities provided by the sensor nodes via web services. The major aim of the CoBIs middleware is to mediate service requests between the application layer and the device layer. The focus lies thereby on deployment and discovery of required services.

AGIMONE [12] is a middleware solution supporting the integration of WSNs and IP networks. It focuses on the distribution and coordination of WSN applications across WSN boundaries. AGIMONE integrates the AGILLA [10] and LIMONE [9] middleware platforms. AGIMONE is a general-purpose middleware with a uniform programming model for applications, that integrates multiple WSNs and the IP network. In our approach, we also promote the integration of sensor nodes via *software connectors*. Moreover, we enable spontaneous communications with some sensor nodes that execute a lightweight version of DIGIHOME.

5.4 Enterprise Service Bus

Enterprise Service Bus (ESB) is an architectural style that leverages on the integration of business infrastructures. In particular, platforms conforming to the *Java Business Integration* (JBI) specification [34] define the concept of *Binding Component* to deal with the heterogeneity of communication standards by exposing the integrated services as WSDL interfaces. The DIGIHOME platform rather exploits the concept of *software connector* to expose the information available in the surrounding environment as REST resources. This data orientation leverages on the integration of smart home devices and contributes to the reactivity of the system by introducing a reasonable overhead compared to more traditional RPC-based protocols.

6 Conclusions and Future Work

In this paper we have presented DIGIHOME, a platform to deal with the mobility, heterogeneity, and adaptation issues in smart homes. In particular, DIGIHOME enables the integration of context information by defining intermediaries that follow REST principles and identifies adaptation situations using this information. The simplicity and data orientation promoted by REST makes it an attractive alternative solution to deal with heterogeneity in terms of interactions. The software connectors of DIGIHOME also enable spontaneous communications by supporting standard protocols (*e.g.*, UPnP and SLP) and furnishing context provider selection (based on QoC attributes). On the other hand, the modularized architecture of DIGIHOME allows the definition of variants for the platform that can be deployed on resource-constrained devices. Furthermore, the clear separation of concerns in the DIGIHOME architecture encourages the exploitation of WSNs for simple processing and local decision making. The suitability of our platform for context integration was evaluated with different discovery and context representations.

Future work includes the integration of our platform with FraSCAti [28], a platform conforming to the SCA specification [24]. By integrating our approach with SCA, we foster the reuse of the different components of DIGIHOME's architecture and support the use of different technologies for the implementation of its components. Furthermore, we can benefit from FraSCAti's reconfiguration capabilities and the separation of concerns of SCA to integrate new communication and discovery protocols at runtime.

Acknowledgement. This work is partly funded by the EGIDE Aurora and INRIA SeaS research initiatives.

References

1. Zigbee Alliance. ZigBee and Wireless Radio Frequency Coexistence. `http://www.zigbee.org/imwp/download.asp?ContentID=11745`, june 2007.
2. Éric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL component model and its support in Java. *Software: Practice and Experience – Special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, August 2006. John Wiley & Sons.
3. Gavin E. Churcher and Jeff Foley. Applying and extending sensor web enablement to a telecare sensor network architecture. In *COMSWARE '09: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, pages 1–6, New York, NY, USA, 2009. ACM.
4. COBIS Consortium. Cobis. fp strep project ist 004270, 2009. http://www.cobis-online.de.
5. Ivica Crnkovic. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.
6. Entertainment Services and Technology Association (ESTA). Architecture for Control Networks (ACN). `http://www.engarts.eclipse.co.uk/acn/`.
7. EsperTech. Esper. `http://esper.codehaus.org/`.
8. Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
9. C.-L. Fok, G.-C. Roman, and G. Hackmann. A lightweight coordination middleware for mobile computing. In *Coordination 2004 : Proceedings of the 6th International Conference on Coordination Models and Languages*, pages 135–151. Springer-Verlag, 2006.
10. L. Fok, G.-C. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. In *IPSN'05 : Proceedings of the International Conference on Information Processing in Sensor Networks*. IEEE, 2006.
11. E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard). `http://tools.ietf.org/html/rfc2608`, june 1999.
12. G. Hackmann, C.-L. Fok, G.-C. Roman, and C. Lu. Agimone: Middleware support for seamless integration of sensor and ip networks. In *DCOSS '06: International Conference on Distributed Computing in Sensor Systems*. Springer, 2006.
13. Karen Henricksen, Jadwiga Indulska, and Ted Mcfadden. Middleware for Distributed Context-Aware Systems. In *International Symposium on Distributed Objects and Applications (DOA'05)*, pages 846–863. Springer, November 2005.
14. Xiaoming Hu, Yun Ding, Nearchos Paspallis, Pyrros Bratskas, George A Papadopoulos, Paolo Barone, and Alessandro Mamelli. A Peer-to-Peer based infrastructure for Context Distribution in Mobile and Ubiquitous Environments. In *Proceedings of 3rd International Workshop on Context-Aware Mobile Systems (CAMS'07)*, Vilamoura, Algarve, Portugal, November 2007.
15. IBM. Web Intermediaries (WIB). `http://www.almaden.ibm.com/cs/wbi/`.
16. Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81, 2002.
17. M. Kirsch-Pinheiro, Y. Vanrompay, K. Victor, Y. Berbers, M. Valla, C. Frà, A. Mamelli, P. Barone, X. Hu, A. Devlic, and G. Panagiotou. Context Grouping Mechanism for Context Distribution in Ubiquitous Environments. In *Proceedings of the OTM International Conferences on Distributed Objects and Applications (DOA'08)*, LNCS, pages 571–588, Monterrey, Mexico, November 2008. Springer.

18. Michael Krause and Iris Hochstatter. Challenges in Modelling and Using Quality of Context (QoC). In *Proceedings of the 2nd International Workshop on Mobility Aware Technologies and Applications*, pages 324–333, Montreal, Canada, 2005.
19. David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
20. Kevin Makice. *Twitter API: Up and Running Learn How to Build Applications with the Twitter API*. O'Reilly Media, Inc., 2009.
21. Nokia. Mobile Web Server, 2008. `http://wiki.opensource.nokia.com/projects/Mobile_Web_Server`.
22. Russel Nzekwa, Romain Rouvoy, and Lionel Seinturier. Towards a Stable Decision-Making Middleware for Very-Large-Scale Self-Adaptive Systems. In *BENEVOL 2009 : The 8th BElgian-NEtherlands software eVOLution seminar*, 2009.
23. Russel Nzekwa, Romain Rouvoy, and Lionel Seinturier. A Flexible Context Stabilization Approach for Self-Adaptive Application. In *Proceedings of the 7th IEEE Workshop on Context Modeling and Reasoning (CoMoRea)*, page 6, Mannheim, Germany, March 2010.
24. Open SOA. Service Component Architecture Specifications, November 2007. `http://www.osoa.org/display/Main/Service+Component+Architecture+Home`.
25. OSGi Alliance. OSGi- The Dynamic Module System for Java. `http://www.osgi.org`.
26. Federica Paganelli, Gabriele Bianchi, and Dino Giuli. A Context Model for Context-Aware System Design Towards the Ambient Intelligence Vision: Experiences in the eTourism Domain. In *Universal Access in Ambient Intelligence Environments*, pages 173–191, 2006.
27. P. Saint-Andre. RFC 3920 - Extensible Messaging and Presence Protocol (XMPP): Core. `http://tools.ietf.org/html/rfc3920`, January 2004.
28. Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and Jean-Bernard Stefani. Reconfigurable sca applications with the frascati platform. In *SCC '09: Proceedings of the 2009 IEEE International Conference on Services Computing*, pages 268–275, Washington, DC, USA, 2009. IEEE Computer Society.
29. Carl-Fredrik Sorensen, Maomao Wu, Thirunavukkarasu Sivaharan, Gordon S. Blair, Paul Okanda, Adrian Friday, and Hector Duran-Limon. A context-aware middleware for applications in mobile Ad Hoc environments. In *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC'04)*, pages 107–110, Toronto, Ontario, Canada, October 2004. ACM.
30. R. N. Taylor, Nenad Medvidovic, and Irvine E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, January 2009.
31. The Apache Software Foundation. HTTP Server Project. `http://httpd.apache.org`.
32. S. Tilak, N.B. Abu-Ghazaleh K. Chiu, and T. Fountain. Dynamic Resource Discovery for Wireless Sensor Networks, 2005.
33. UPnP Forum. UPnP Device Architecture 1.0. `http://www.upnp.org/resources/documents.asp`, april 2008.
34. Steve Vinoski. Java Business Integration. *IEEE Internet Computing*, 9(4):89–91, 2005.
35. Guangming Wang and Gonglian Jin. Research and Design of RFID Data Processing Model Based on Complex Event Processing. In *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 1396–1399, Washington, DC, USA, 2008. IEEE Computer Society.
36. Weixin Wang, Jongwoo Sung, and Daeyoung Kim. Complex event processing in epc sensor network middleware for both rfid and wsn. In *ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, pages 165–169, Washington, DC, USA, 2008. IEEE Computer Society.
37. Zigbee Alliance. Zigbee Protocol. `http://www.zigbee.org/`.