Journal of Computer Science

Original Research Paper

Using MVCA to Improve Architecture Modularity of Smart Spaces

[1]Somia Belaidouni, [2]Moeiz Miraoui and [1]Chakib Tadj

[1]École de Technologie Supérieure, Montreal, Canada
[2]High Institute of Applied Sciences and Technology, University of Gafsa, Tunisia

Article history
Received: 06-03-2017
Revised: 28-08-2017
Accepted: 20-10-2017

Corresponding Author:
Somia Belaidouni
LATIS Laboratory, École de Technologie Supérieure, Montreal, Canada
Email: belaidounisoumia@yahoo.fr

Abstract: There has been increasing interest in the use of context awareness, as a technique for designing architectures dedicated to smart spaces in order to adapt and produce suitable services according to user context. In recent years, various architectures have been developed to support context-aware systems. The major challenge with these systems is decomposing the entire architecture into smaller, modular components that facilitate the comprehension and modification of the architecture. In this study, we propose the Model View Controller Adapter (MVCA) architecture, derived from the model-view-controller pattern, which is modular, flexible and capable of adapting services autonomously on behalf of users. The main concept of MVCA architecture is that it decomposes the overall functionalities into modular components with high cohesion and low coupling, which facilitates reusability and maintainability of the system. The MVCA architecture is essentially composed of four components that are responsible for sensing and managing the environmental context in order to adapt and produce services proactively according to user context. To clarify and show the usability of our architecture, we present a scenario-based simulation of MVCA architecture using the Java Agent Development Framework platform.

Keywords: Pervasive Computing, Smart Space, Architecture, Context-Awareness, MVC Pattern, Modularity

Introduction

The diverse nature of pervasive computing makes it difficult for software designers to adopt one common model that can meet all requirements (Abdualrazak *et al*., 2010). Designing context-aware smart spaces is a challenging task for two main reasons: Firstly, supporting different devices and multiple interacting platforms is difficult; and secondly, it is difficult to achieve environmental context awareness and to proactively adapt services to dynamic changes. Addressing these issues means that systems should be capable of providing a uniform and efficient architecture for communicating the entirety of entities within the environment to meet people's needs and adapt to their preferences.

In view of these requirements, our project aims to create a modular and flexible architecture, which takes the context as the first input as the most important element for reasoning, adapting and providing a service in a more suitable form. Modularity is a significant quality attribute in the design of large system architectures (Knoernschild, 2012). Modularity is considered as a key feature to improve several quality attributes such as maintainability, portability, reusability, interoperability and flexibility (Galin, 2004). In the case of our smart environment, modularity is the key to facilitating management of the design complexity and ensuring the development of the overall components. Moreover, the degree of modularity is proportional to the degree of loose coupling and high cohesiveness of a system's software elements (Sharafi *et al*., 2012). Therefore, the benefit of modularity is dividing the architecture into loosely coupled and highly cohesive modules, which interact in a collaborative manner to achieve the requirements.

The Model-View-Controller (MVC) is considered as an architectural pattern in software engineering. It adopts the idea of "divide and conquer," (Zhang and Zhu, 2013), which involves dividing the entire

architecture into three separate components: Model, view and controller, based on the principle of separating the act controllers, data presentation and user's actions. This facilitates modification and change in each component without significantly disturbing the others. The benefits of the MVC design have been demonstrated in various interactive applications (Sarker and Apu, 2014).

In this study, we propose a Model View Controller Adapter (MVCA) architecture that is a variation of the established MVC pattern and provides a new perspective on modularization. It divides the overall functionalities of the system into a modularized architecture. MVCA introduces a new component, the adapter, to provide suitable services with a robust and extensible infrastructure in order to provide environmental context awareness and meet the user's needs. Mainly, it is responsible for reasoning regarding the sensed context and adapting services proactively. In general, the adapter is a set of rules which have to be specified for the possible contextual configurations about user interactions and their preferences and each rule triggers a specific set of services according to the current context. In such adapter, we have to imagine and predict all possible context configurations before deploying the system. This set of rules will then be static and can not be modified when the system is operational. Such approach has two main drawbacks: (a) it is impossible to predict all possible context configurations and (b) the adaptation system is not dynamic and cannot evolve as the system operates (it is limited to the static set of rules). Instead of rule base approach, the component contains also learning engine in order to adapt services to all possible and meaningful context configurations. This machine learning approach can select a previous choice about the service and can adapt itself by another new choice about service from user feedback, which make it adaptative.

The remainder of this paper is organized as follows. In the next section, we provide an evaluation of some existing software architectures. In section 3, we describe our proposed architectural pattern to overcome the identified problems. In section 4, we present a scenario implementation and report the main results from the simulation studies. Finally, our conclusion and future work are presented in section 5.

Related Work

A wide variety of context-aware schemes have been proposed over the past few years. However, most of these provide only partial context-awareness functionality (Zhang *et al*., 2013) and aim to satisfy certain quality attributes, such as interoperability, flexibility and maintainability. The CASS tool (Clarke and Fahy, 2004) is middleware for supporting the development of context-aware applications. It divides the entire architecture into independent components and achieves effective abstraction of contextual information. This architecture provides efficient modularity that facilitates the modification of server components; in particular, the inference engine. Octopus (Firner *et al*., 2012) is dynamically extensible middleware for facilitating smart home/office domain-specific applications. It is constructed on a simple layered architecture to tackle the problems of data management and fusion; however, it appears to have a lower tolerance for problems. FlexRFID (El Khaddar *et al*., 2015) aims to provide a policy-based middleware solution for facilitating the development of context-aware applications and integrating heterogeneous devices. Authors have furthermore proposed a multi-agent (Miraoui *et al*., 2016) for a smart living room that focuses on context-awareness, which is a key enabler for service adaptation in smart spaces. The project aims to provide architecture with a high degree of modularity; that will have a positive impact on several software qualities, such as modifiability, reusability, maintainability and extensibility. The architecture proposed in (Aloulou *et al*., 2012) provides an adaptable and dynamic platform that integrates new assistive services at runtime and enables their binding to specific patients. This system adopts modularity by using the Service-Oriented Approach (SOA), which improves its flexibility, scalability and configurability. The FIWARE (FIWARE Project, 2016) aims to provide cutting-edge infrastructure in which the creation and delivery of services, high quality of service and effective security are achieved. It is a generic platform that could adaptively be applied to various usage areas. Preuveneers and Berbers (2007), authors have proposed a resource-aware and context-driven application middleware for mobile devices that has a layered design and provides introspection and intercession capabilities within each layer to support a more flexible self-adaptation strategy at runtime. The MUSIC project (Rouvoy *et al*., 2009) is a middleware Support for Self-Adaptation in ubiquitous and service-oriented environments, which addresses aspect-oriented adaptations. To accomplish this, authors uses computations and evaluations of alternative application configurations in response to context changes and the selection of a feasible configuration. The SECAS (Chaari *et al*., 2008) is a project that ensures the deployment of adaptive context-aware applications. The authors aim to develop a platform, which makes the services, data and the user interface of applications adaptable to different context situations. Focuses on this point by providing the necessary tools to adapt existing applications to new contextual situations that were not taken in consideration at their design stage. The Chisel system (Keeney and Cahill, 2003) is an open framework for dynamic adaptation of services using reflection in a policy-driven, context-aware manner. The system is based on decomposing the particular aspects of a service object that do not provide its core functionality into

multiple possible behaviours. The principal aim of this project is to build a framework supporting unanticipated dynamic adaptation that will take account of contextual information from as many sources as possible. The PACE middleware (Henricksen *et al*., 2005) is used to support the development of a context-aware vertical handover application which investigates a variety of issues related to pervasive computing, including the design of context-aware applications and solutions for modeling and managing context information. Il offers a toolkit that facilitates interaction between application components and the context and preference management systems.

Overview of MVCA

The MVCA architecture is based on the MVC pattern, with an additional component, namely the adapter, which is used to adapt services in unforeseen situations. This architecture is modular and based on the separation of the four components that represent the functionalities of a smart space. These modules interact and collaborate to handle contextual data in order to produce the most suitable service.

The context is the key element to take into account because it represents the main input data on which our architecture functionalities are based. As illustrated in Fig. 1, we have decomposed the smart space into three main parts: The context is represented by the contextual model, the adaptation part is realized by the adapter and controller and the service part is represented by the view component.

Detailed MVCA Architecture

In terms of software, the system consists of four main components: The controller, contextual model, adapter and view, as shown in Fig. 2. (1) The controller receives a variety of contexts sensed from the environment. The context is then interpreted in a suitable form to be comprehensible and used by other components. (2) The contextual model represents the storage of all context information. This component contains sensed data associated with the appropriate services, as well as a pre-installed list of user preferences. Moreover, it is responsible for responding to controller requests by sending the appropriate services, if they are found. It can also receive new services that have been adapted by the adapter component due to unforeseen circumstances. (3) The adapter module is the core component of the architecture, as it maintains the overall mechanisms and rules to adapt services to new situations. Firstly, the adapter receives contexts from the controller and attempts to reason and adapt suitable services. Then, it sends the adapted services, accompanied by their context information, to the contextual model, to be saved for future utilization. (4) Finally, the view component is responsible for determining the system output. It receives queries from the adapter to execute services such as displaying data or requests sent to actioners.

Controller

The controller is the first component to interact with the environment. Thus, it is related to sensors and devices implanted in the environment in order to intercept and collect any context information or situation changes. This component is the backbone of the architecture because it provides the main support for context to be used by others components. Therefore, it includes procedures for interpreting and reasoning, in order to transform gathered low-level raw data into higher- level information. The controller component performs three major interactions:

- Transfers the appropriate service from the contextual model component to the actual context
- If the service is found, sends the service in question to the view component for execution
- Otherwise, sends the context information to the adapter component

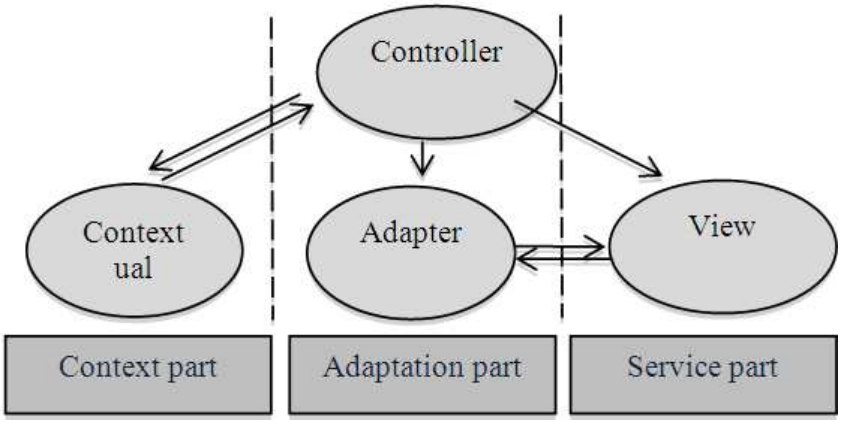

Fig.1. MVCA architecture

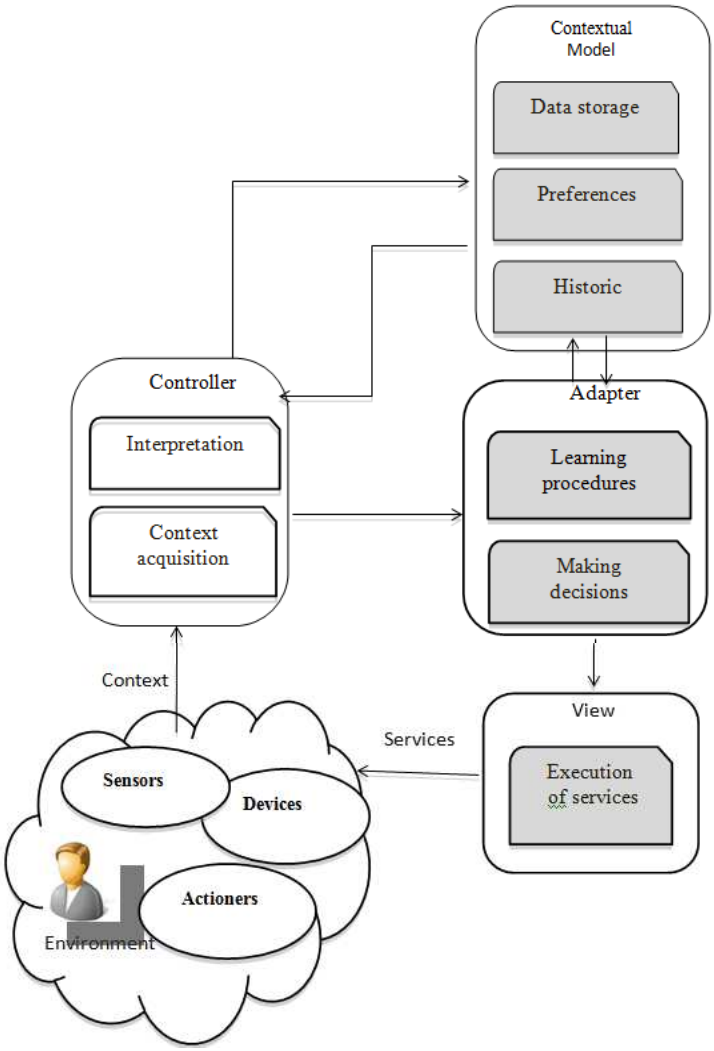

Fig. 2. MVCA components

Contextual Model

The contextual model acts as data storage for all contexts collected by the controller component. This component incorporates a context history accompanied by particular service criteria that may be used to establish trends and predict future services. It also saves also users' information and preferences in order to satisfy their needs. The contextual model is responsible for the following interactions:

- Checks in the base if there is a match between the context received from the controller and the recorded context
- Sends a message response to the controller component containing the appropriate service to execute (if found), else sends a negative response
- Saves the adapted services with their context information received from the adapter component

Adapter

The adapter component contains the domain-level logic of the system. It holds a database of adaptation rules. This rules define the adaptation operations that the adapter has to compute on the service it adapts and the context parameters related to it.

Besides, it is based on user's behavior learning using different types of procedures for automatic learning of the best possible services to execute, as well as a decision-making process, which is responsible for determining which service should be performed in a given situation in order to provide personalized and relevant information to end-user. Then, mainly this process involves two general requirements that form cycle. First gathering preferences of the user (with feedback or history of interaction for instance) and then adapting the engine to produce a suitable service (for instance by adjusting parameters related to preferences). For example, an inhabitant that gets up

generally at 7 in the morning. One day, since he has a meeting in another city, his alarm rings at 6 'o'clock because the inhabitant set the adaptation system on the adaptation server before he went to sleep. Based on this example, we deduce that. That is why the system must be aware to the context and react to changes of this context in a continuous way. In order to get the right context of the user and his environment to fulfill adaptation process, the adapter component continuously interacts with the other components to perform the following operations:

- Receives the context information from the controller component
- Adapts suitable services according to situation changes
- Retrieves useful information during the adaptation process as required
- Sends adapted services that are found, along with their context information, to the contextual model to be saved for future use
- Sends adapted services to the view component for execution

When the adaptation process was performed and the service is executed, the user must have the option to accept or reject the service providing a feedback that will be used to adjust the parameters of the techniques implemented.

View

The view component aims to generate and provide services to the user. It mainly contains mechanisms for interacting with actuators, which perform services at the final stage. These services are in the form of notifications or recommendations forwarded to user devices and can also be queries performed by actuators to execute services. The view component is responsible for the following operations:

- Receives services from the adapter component
- Sends notifications or recommendation to user devices
- Sends queries to actuators to perform received services

In this study, the MVCA architecture is implemented using the Java Agent Development Framework (JADE) platform (Bellifemine *et al*., 2005). Figure 3 shows the four different types of agents representing the MVCA architecture components. It should be noted that we have added a new agent, "context," which represents the context information sensed from the environment.

The interaction between different agents follows a process, consisting of the following phases:

- When a contextual agent obtains external information, it informs the controller agent of the available context in order to find the appropriate service
- The controller agent reads and processes the received information and requests the appropriate service from the contextual model
- When the contextual model receives context information, it checks its base to determine whether there is a match between the received and saved contexts. A negative response is sent when no match is found and a positive response when a match between service and context elements is identified.
- If the service is identified by the contextual model agent, the controller agent sends a query to the view agent with the service's criteria in order for
- It to be executed, as an appropriate service for user needs
- If the service is not identified, the controller agent informs the adapter agent about the identified context requiring an adaptation service
- The adapter agent uses mechanisms to adapt and determine suitable services, which are then transferred to the view agent. Moreover, it communicates the new adapted service to the contextual model agent according to its context information, to be added to the database for future work
- The adapter agent retrieves certain information from the model contextual agent that will be used in learning processes, such as historical or preference data, in order to determine the appropriate service.
- Once the view agent receives the specific service, it responds to orders received from the adapter agent and sends notifications, recommendations, or queries to actioners installed in the environment.

In order to verify the performance of the MVCA architecture, we implement a simple scenario using JADE agents. The scenario describes the daily routine of an employee in his office from 8:00 a.m. until 4:00 p.m.

The entire scenario can be divided into five scenes. The basic concept of each scene is that the context agent is the first to sense the context that will be interpreted and processed by the controller agent in order to infer the appropriate action to be taken, either directly by the contextual model agent or by the adapter agent. Then, the view agent receives the services to be executed. Parameters reflecting all contextual variation in the environment are taken into account; for example, working day, time, place, temperature, light and device interaction.

JADE Agents

The scenario describes the daily routine of an employee in his office from 8:00 a.m. until 4:00 p.m.

The entire scenario can be divided into five scenes. The basic concept of each scene is that the context agent is the first to sense the context that will be interpreted and processed by the controller agent in order to infer the appropriate action to be taken, either directly by the contextual model agent or by the adapter agent. Then, the view agent receives the services to be executed. Parameters reflecting all contextual variation in the environment are taken into account; for example, working day, time, place, temperature, light and device interaction.

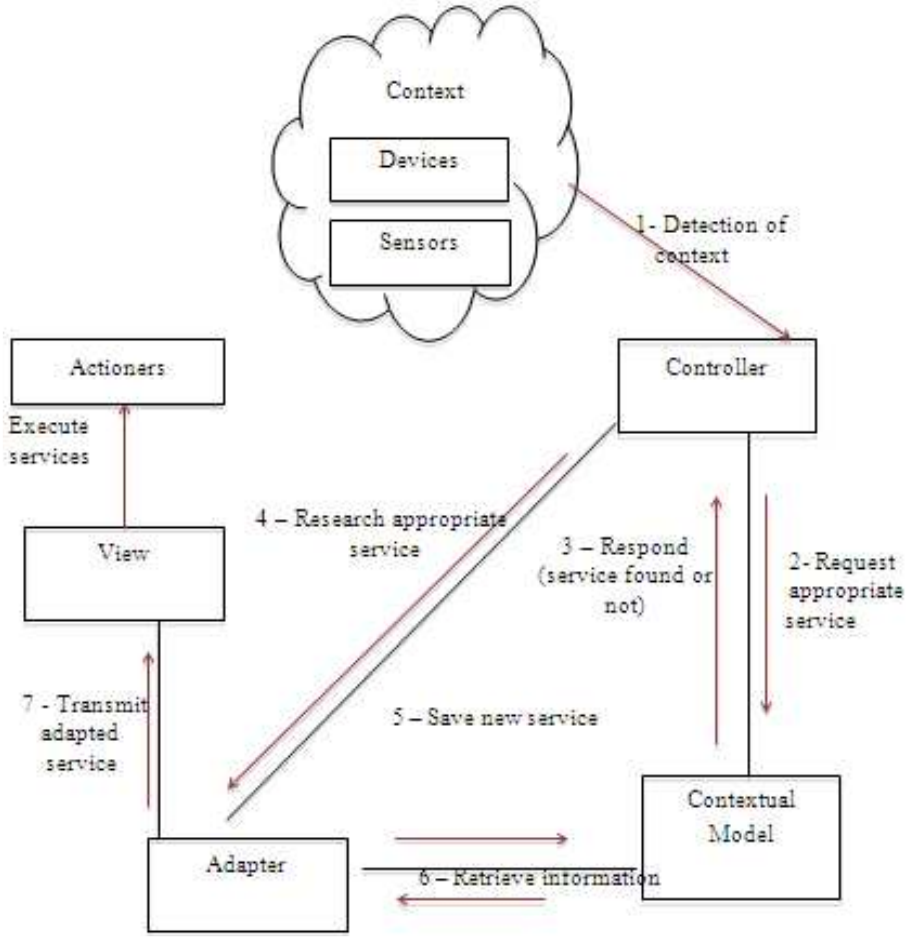

Fig. 3. Interaction agents in MVCA architecture

Table 1 illustrates the different scenes, each of which describes a particular service. We choose to identify scenes according to the three most important pieces of information: (1) contextual data, which can either be planned situations in the system or unanticipated situations requiring an adaptation process; (2) the equipment performing the action; and (3) the performed service, identified by its nature, which can be either a default service (predefined service according to a pre-established context in the system) or adapted service (new service resulting from an adaptation process in an unanticipated situation).

We take the third and fourth scenes, representing two different situations, as an example. The first is a planned situation; that is, a situation with a known service. The sensed context relates to the day (working day), time (8:00 a.m.) and interaction of equipment (door). This information is identified as the main input of the MVCA architecture, to be transmitted to the controller component, which in turn attempts to retrieve the appropriate service from the contextual model module. In response, the contextual model sends an adequate service description located in its local database. This is based on the fact that, in the case of planned situations, the contextual model component has already saved different context information, such as preferences or known conditions associated with suitable services, to be executed in the future. Finally, the service will be transmitted directly to the view component, which is responsible for sending orders to turn the light on.

The second scene involves unanticipated situations that have not been previously saved in the system. The context agent senses that the conference room is occupied, a situation that is not predefined in the system.

Therefore, the controller agent receives a negative response (service not found) from the contextual model.

The controller then communicates this context information to the adapter, which can reason and adapt a suitable service; in this case, a notification about changing the conference room.

Accordingly, the view sends notifications to the user's cell phone to inform them about the alternative room where the conference can be held. Table 2 illustrates different interactions between agents in the two scenes.

Table 1. Description of scenario scenes

	Contextual data			Executed service	
	Planned situation	Unanticipated situations	Equipment	Nature	Description
Scene 1	Working day Time: 8:00 a.m.		Light system	Request sent to lamps (Default service)	Lamps are turned on
Scene 2	Door opens Conference room: A50	Alternative room: B30	Cell phone	Notification (Adapted service)	Message indicating the new conference room
Scene 3	Ali leaves his office		Portable PC Light system	Requests sent to portable PC and light system (Default service)	PC and light system enter power save mode
Scene 4	Ali returns to the office	The temperature is higher than 25 degrees	Air-conditioning system	Request sent to air -conditioning system (Adapted service)	Air conditioning turned on to reach the desired temperature
Scene 5	Time: 4:00 p.m. Door closes Ali lefts his office		Portable PC Light system Air-conditioning system	Requests sent to equipment (Default service)	Equipment is switched off

Table 2. Interactions between agents in two different scenes

Scene	Context	A. Ctxt	A. Contr	A. MDL	A. Adap	A. View	Query
Scene 1	Working day	E	R				Inform
	Time: 8:00 a.m.		E	R			Request
	Door opens		R	E			Respond (Service found)
			E			R	Execute service
Scene 2	Conference room occupied	E	R				Inform
			E	R			Request (Service not found)
			E	R			Respond
			E		R		Request
				R	E		Respond (adaptable service)
					E	R	Execute service

We now discuss the implementation results. We developed a simple application to test our architecture. Fig. 4 displays a snapshot of the application interface, which is designed to load a scenario in a text format and launch the creation of agents for the simulation. Furthermore, the interface shows different scenes extracted from the associated scenario, with the elapsed time between the start and completion of each scene. The interface also presents an automat diagram that shows the various interactions between agents in each scenario. An agent that is known to simply receive a message is colored red. This explains the communication flow among agents in the MVCA architecture.

The sequence diagram shown in Fig. 5 depicts the scenario execution, which is carried out by the exchange of messages between different agents composing the MVCA architecture. The diagram is also used to show how the different components of our architecture cooperate to accomplish the proposed scenes.

The diagram illustrates the occurrence of events for invoking specific operations using different behavior messages like “inform, request.” Execution times, sender, location and context are annotated in the sequence diagram, as shown in Fig. 6.

In this system, modularity is realized by separating the MVCA functionality into independent parts, so that each module focuses only on one or a small number of interactions. This method contributes to maximizing the interaction within each component (cohesion) and minimizing dependencies (coupling) between components. It also facilities the modification of components to satisfy local requirements. Moreover, it contributes to reducing system maintenance efforts and maximizing the system’s reusability.

Figure 6 shows four tracking tables for the

components, and shows interactions during the pre-established scenes at each instant. For example, in the controller table report, the first interaction shows that at 5:40:12, the controller component receives a message from the context agent with sened information (working day, time: 8:00 a.m., door opens), and the user location is "office." This context is transferred to the contextual model agent at 5:40:14 to determine the appropriate service (turn on lamps), which is executed by the view agent at 5:40:20.

The execution times of the different component interactions components highlight the progress of the MVCA architecture. Moreover, we can retrace errors that are a result of poor component operation.

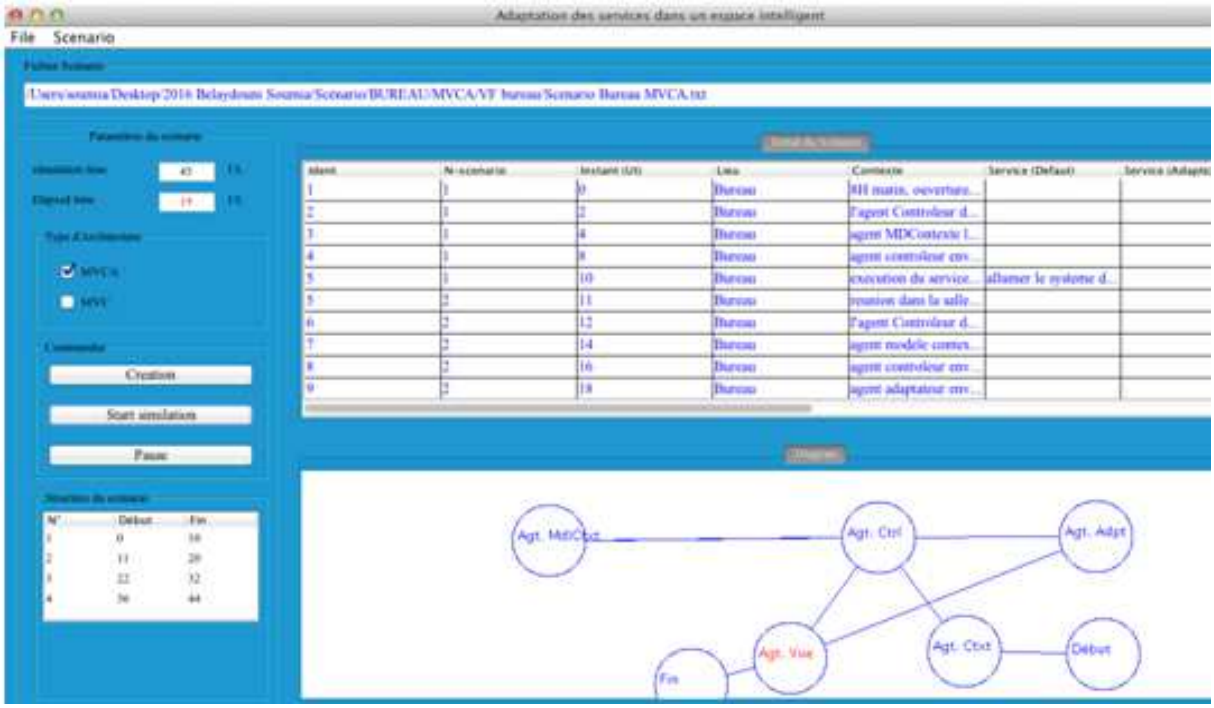

Fig. 4. Interface of MVCA implementation

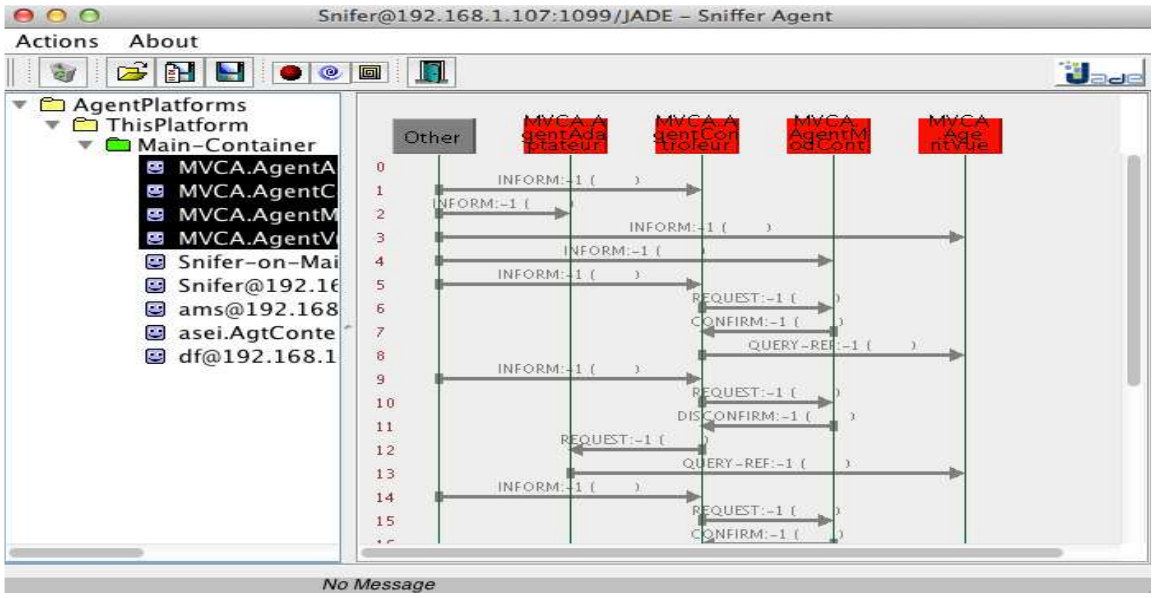

Fig. 5. Sequence diagram for MVCA architecture agent interactions

■■■

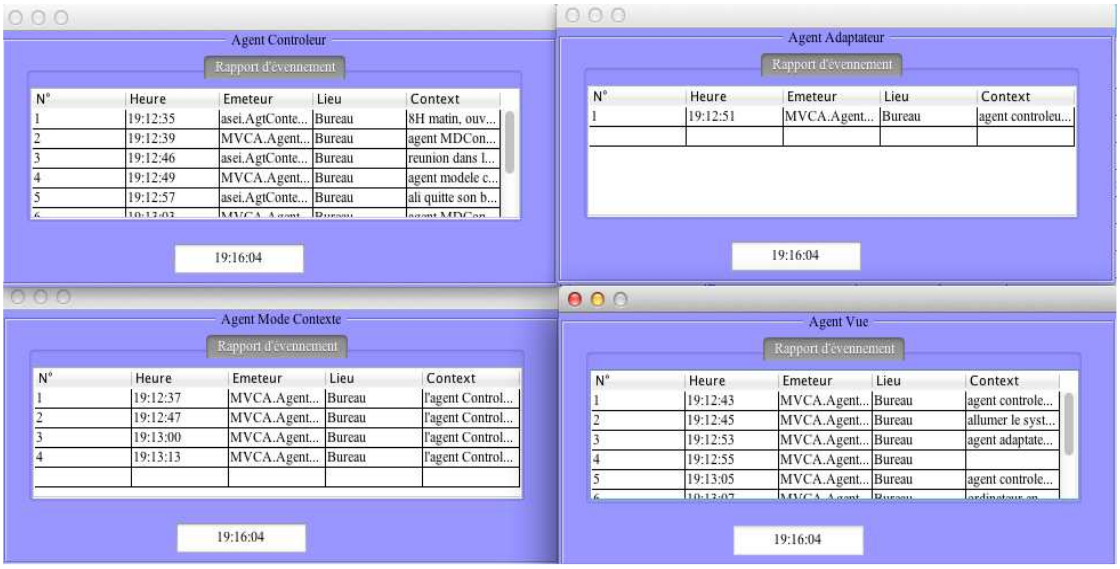

Fig. 6. Component event report

Conclusion

Smart environments are characterized by complex systems that require a balance between transparency and context awareness. Architectures for such systems must respond to requirement space and should integrate a modular and flexible design that can produce appropriate services at the right time. In this article, we propose MVCA architecture, derived from the MVC pattern. The main idea behind this system is to decompose the overall functionalities of into four separate components, which are responsible for sensing, managing context information and reasoning to execute the most suitable service for users. This approach allows for high cohesion and low coupling, which are important measures of architecture quality. Therefore, this method facilitates system reusability and maintainability. In future works, we plan to extend the MVCA architecture to handle and process various complex scenarios and to evaluate the performance of the architecture and its implementation in depth.

Acknowledgement

This work has been achieved at the École de technologie supérieure (Montréal, Canada).

The author would like to thank everyone who has contributed to the progress of this research.

Author's Contributions

Somia Belaidouni: The development of the Model View Controller Adapter (MVCA) architecture for smart spaces.

Moeiz Miraoui: Simulation and evaluation of the proposed architecture.

Chakib Tadj: The state of the art and work supervision.

Ethics

This article is the original contribution of the authors and is not published elsewhere. There is no ethical issue involved in this article.

References

Abdualrazak, B., Y. Malik and H.I. Yang, 2010. A taxonomy driven approach towards evaluating pervasive computing system. Proceedings of the International Conference on Smart Homes and Health Telematics, (HHT'10), Springer Berlin Heidelberg, pp: 32-42, DOI: 10.1007/978-3-642-13778-5_5

Aloulou, H., M. Mokhtari, T. Tiberghien, J. Biswas and P. Yap, 2014. An adaptable and flexible framework for assistive living of cognitively impaired people. IEEE J. Biomed. Health Informat., 18: 353-360. DOI: 10.1109/JBHI.2013.2278473

Bellifemine, F., F. Bergenti, G. Caire and A. Poggi, 2005. Jade - A Java Agent Development Framework. In : Multi-Agent Programming, Bordini, R.H., M. Dastani, J. Dix and A. El Fallah Seghrouchni (Eds.), Springer, Boston, MA, ISBN-10 : 978-0-387-24568-3, pp: 125-147.

Chaari, T., F. Laforest and A. Celentano, 2008. Adaptation in context-aware pervasive information systems: The SECAS project. Int. J. Pervasive Comput. Communications, 3: 400-425. DOI : 10.1108/17427370710863130

Clarke, P. and S. Fahy, 2004. CASS-middleware for mobile context-aware applications. Proceedings of the Workshop on Context Awareness, (WCA'04), CiteSeerX, pp: 1-6.

El Khaddar, M.A., M. Chraibi, H. Harroud, M. Boulmalf and M. Elkoutbi *et al*., 2015. A policy-based middleware for context-aware pervasive computing. Int. J. Pervasive Comput. Communi., 11 : 43-68.

Firner B., R.S. Moore, R. Howard, R.P. Martin and Y. Zhang, 2011. Smart buildings, sensor networks and the internet of things. Proceedings of the 9th Conference on Embedded Networked Sensor Systems, Nov. 01-04, ACM, New York, USA, pp: 337-338. DOI: 10.1145/2070942.2070978

FIWARE Project, 2016. https://www.fiware.org/

Galin, D., 2004. Software Quality Assurance: From Theory to Implementation. 1st Edn., Pearson Education Limited, Harlow, ISBN-10: 0201709457, pp: 590.

Henricksen, K., J. Indulska, T. McFadden and S. Balasubramaniam, 2005. Middleware for distributed context-aware systems. Proceedings of the OTM Confederated International Conferences on the Move to Meaningful Internet Systems, Oct. 31-Nov. 04, Springer, Berlin, Heidelberg, Agia Napa, Cyprus, pp: 846-863. DOI: 10.1007/11575771_53

Keeney, J. and V. Cahill, 2003. Chisel: A policy-driven, context-aware, dynamic adaptation framework. Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks, Jun. 4-6, IEEE Xplore Press, Lake Como, Italy, pp: 3-14. DOI : 10.1109/POLICY.2003.1206953

Knoernschild, K., 2012. Java Application Architecture: Modularity Patterns with Examples Using OSGi. 1st Edn., Prentice Hall Press, ISBN-10: 0321247132, pp: 384.

Miraoui, M., S. El-etriby, A.Z. Abid and C. Tadj, 2016. Agent-based context-aware architecture for a smart living room. Int. J. Smart Home, 10: 39-54. DOI: 10.14257/ijsh.2016.10. 5.05

Preuveneers, D. and Y. Berbers, 2007. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. Proceedings of the Symposium on Applied Computing, Mar. 11-15, ACM, Seoul, Korea, pp: 1165-1170. DOI: 10.1145/1244002.1244255

Rouvoy, R., P. Barone, Y. Ding, F. Eliassen and S. Hallsteinsen *et al*., 2009. Music: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. In: Software Engineering for Self-Adaptive Systems, Cheng, B.H.C., R. de Lemos, H. Giese, P. Inverardi and J. Magee (Eds.), Springer Berlin Heidelberg, ISBN-10: 978-3-642-02161-9, pp : 164-182.

Sarker, I.H. and K. Apu, 2014. MVC architecture driven design and implementation of java framework for developing desktop application. Int. J. Hybrid Inform. Technol., 7: 317-322. DOI: 10.14257/ijhit.2014.7.5.29

Sharafi, S.M., M. Ghasemi and N. Nematbakhsh, 2012. Using architectural patterns to improve modularity in software architectural design. Proceedings of the International Conference on Software and Computer Applications, (SCA' 12), IACSIT Press, Singapore, pp: 65-70.

Zhang, D., H. Huang, C.F. Lai, X. Liang and Q. Zou *et al*., 2013. Survey on context-awareness in ubiquitous media. Multimedia Tools Applic., 67: 179-211. DOI: 10.1007/s11042-011-0940-9

Zhang, H. and S. Zhu, 2013. B/S implementation of internet-based electrical engineering lab with MVC architecture. Proceedings of the 10th IEEE International Conference on Control and Automation, Jun. 12-14, IEEE Xplore Press, Hangzhou, China, pp: 551-555. DOI: 10.1109/ICCA.2013.6564862