AjTempura – First Software Prototype of C3A Model

Vladimir Valkanov[1], Asya Stoyanova-Doycheva[1], Emil Doychev[1], Stanimir Stoyanov[1], Ivan Popchev[2], and Irina Radeva[2]

[1] University of Plovdiv, Bulgaria
{stani,astoyanova}@uni-plovdiv.bg,
{vvalkanov,e.doychev}@uni-plovdiv.net
[2] Bulgarian Academy of Science, Bulgaria
{ipopchev,iradeva}@iit.bas.bg

Abstract. The paper provides a general description of a model for context-aware agent architecture (C3A) and first steps in AjTempura creation vie C3A model. The approach adopts the definition of context and context-awareness given by Dey. The C3A model aims at creating of smart virtual spaces. The applicability of the model is demonstrated by development of an agent-oriented application.

Keywords: Context-aware architecture, Formal models, Intelligent agents, Agent-Oriented architectures, Tempura.

2010 Mathematics Subject Classification: 68T42 Agent technology

1 Introduction

One of the main characteristics of the modern systems today (i.e. eLearning) is the 'anytime-anywhere-anyhow' delivery of electronic content, personalized and customized for each individual user. To satisfy this requirements new types of context-aware and adaptive software architectures are needed, which are enabled to sense aspects of the environment and use the information to adapt their behaviour in response to changing situation. From a network of computer networks the Internet has transformed into the Internet of Things [1], where people and everyday objects can be assigned identifiers. These identifiers can be created and managed by computers. Within the infrastructure of the current Web, the new Symantec Web [18] is rising making all sorts of informational resources addressable by the existing identification protocol URI. A model for development of context-aware software architectures, known as Context-Aware Agent Architecture (C3A), is presented in the publication. There are a lot of definition of context-awareness, first attempts to give satisfactory definition are related to application of mobile devices and reading users' position. In the specialized literature it is presented as the term context-aware computing is used to describe software possible to render an account of users' location [10] or as the ability of a program or device to sense various states of its environment and itself [4]. We adopt Dey's context definition for developing C3A model. Dey [3] criticizes the

P. Angelov et al. (eds.), *Intelligent Systems'2014*,
Advances in Intelligent Systems and Computing 322, DOI: 10.1007/978-3-319-11313-5_38

above definitions in two ways: context if defined by examples, i.e. by enumeration of various cases, and context is defined by synonyms, mainly as environments or situations. In his opinion, there definitions should bother development of context-aware applications. In this sense he proposes a more common definition whereby context is any information that can be used to characterize the situation of an entry. An entry can be assumed with a person, a place, or an object that is considered relevant to the interaction between a user and an application, including themselves [10]. In this point of view if a system which provides relevant information and/or services to the user, where relevancy depends on the user's task, is called context-aware system. An implementation of the C3A in real software demonstrating the features of the model is also presented in this paper.

2 C3A Model

The model aims to propose a framework which can be used for implementation of context-aware applications for various domains. A context-aware architecture includes autonomous intelligent components which can: operate in a changing environment; detect, identify and localize changes (events) in the environment; initiate corresponding compensatory actions depending on the types of changes (events). A compensatory action is defined as an activity (functionality) fired as a reaction to the occurrence of some event (change in the environment). Two basic compensatory actions are: *Adaptation* – before the information resources (electronic services, electronic content) are provided, they can be modified so that they reflect as completely as possible the specifics of the change or event; *Personalization* – before the information resources (electronic services, electronic content) are provided, they can be modified so that they reflect as completely as possible the users' desires, intentions, background, etc.

Using C3A a formal presentation is proposed which reflects the spatial and temporal aspects of: virtual space structure and building components; relations between the components; operations allowed in the space; control of the processes taking place in the space.

In the model, the fundamental notion is this of ***smart space***, defined as

$SmartSpace = \bigcup_{i=1}^{p} SP_i$, where there exists a set $SubSpaces = \{SP_i \mid i = 1, \ldots, n\}$ of subspaces and $p \leq n$.

For two subspaces $SP_i \in SubSpaces$ and $SP_j \in SubSpaces$ we introduce the following definitions:

- *Empty subspace*, if $SP_i = \varnothing$;
- *Overlapping subspaces*, if $SP_i \cap SP_j \neq \varnothing$;
- *Identical (or completely overlapping) subspaces*, if $SP_i = SP_j$;
- *Disjunctive subspaces*, if $SP_i \cap SP_j = \varnothing$.

In the smart space, ***services*** will be provided through autonomous intelligent components, known as *agents*. The agents are with 'limited rationality', which in this case means:

- They operate with limited capacity (resources) in order to plan, predict, make choices and execute actions;
- They have partial (local) control and impact on the smart space.

Due to the limited rationality, we would assume that:

- The set of services, provided in the space, alters dynamically;
- There is a *minimal functionality* (minimal set of services), that the space cannot operate without.

Let $Agents = \{a_i \mid i = 1, \ldots, m\}$ be the set of all agents potentially operating in the smart space. Due to the limited rationality we assume that each agent operates in its own subspace (known as *agent's range*) operating as agent's environment. Agents can operate and impact only within their own range. In this sense, the overall smart space (noted as $SmartSpace^A$) is built as union of agents' ranges $R(a_i)$. Furthermore, let $Services = \{s_i \mid i = 1, \ldots, k\}$ be the set of services provided in the $SmartSpace^A$.

Various types of agents and services are distinguished in the $SmartSpace^A$. The set of agents is decomposed as $Agents = PA \cup OA$, so that $PA \cap OA = \varnothing$. Respectively, the set of services can be decomposed as $Services = F \cup AF \cup SF$, i.e. as a union of three mutually disjunctive subsets ($F \cap AF = \varnothing$, $F \cap SF = \varnothing$ and $AF \cap SF = \varnothing$). This decomposition characterizes the following types of services:

- F is the minimal functionality of SmartSpace;
- AF includes the compensatory actions;
- $SF = \{generate, remove, selfremove\}$ includes three special operations which are used to create and remove operational agents. For instance, the generate operation is defined as $generate : PA \times E \rightarrow OA$.

Here, the two types of agents are presented shortly. PA includes agents, known as *persistent agents,* which are always available in the space. These agents are used to:

- Provide the minimal functionality of the space;
- Identify and localize the events which take place in the space;
- Generate operative agents;
- Remove operative agents.

The persistent agents' behavior is defined as the next genetic function

$$Beh_{pa} : PA \rightarrow 2^F \cup \{generate, remove\},$$

where $\{generate, remove\} \subset SF$.

OA includes agents, called *operative agents,* generated dynamically by the persistent agents. An operative agent provides some kind of compensatory actions as reaction to the events identified in the range of the generating persistent agent. After compensatory actions completing, the operative agent will be removed (or self-removed).

The operative agents' behavior is defined as the below genetic function

$$Beh_{oa} : OA \rightarrow 2^{AF} \cup \{selfremove\},$$

where $\{selfremove\} \subset SF$

The events which take place in the space are introduced as *Events* = $\{e_i \mid i = 1, \ldots, e\}$. *Location* and *time of occurrence* are the two basic features of the events. C3A examines the locations of the events' occurrence as unique subspaces, denoted as $R(e_i)$, so that $SmartSpace^E$ presents the overall SmartSpace in terms of the locations of expected events' occurrences. The events are always viewed as *atomic primitives,* and they can be combined to make more complex structures such as situations, scenarios, intervals. An interval is the sequence $EInt = [e_1, \ldots, e_{pr-1}, \boldsymbol{e_{pr}}, e_{pr+1}, \ldots, e_l]$ of events ordered in time where an event e_{pr} can be identified at any time. Event e_{pr} is interpreted as the *present*; the subinterval $[e_1, \ldots, e_{pr-1}]$ is the *past*; respectively, $[e_{pr+1}, \ldots, e_l]$ is the *future*.

As already pointed out, the persistent agents perform two different functions. First, they provide the minimal functionality in the SmartSpace. Second, they can identify the occurrence of certain events in its ranges and dynamically generate appropriate operative agents, respectively, which in turn execute the necessary compensatory actions. In the model, this is formally presented, as follows:

$$\forall a_i \in PA\ (\exists\, a_j \in OA, \exists\, e^* \in E : generate(a_i, e^*) = a_j),$$

under the condition that $R(a_i) \equiv R(e^*)$.

The general lifecycle of a context-aware agent-based software architecture compatible with C3A is shown (in pseudocode) in Fig. 1. Presented in time, the operation of the architecture looks like a pulsating core (implementing the minimal functionality of the SmartSpace) which periodically 'inflates' in different directions (depending on the changes in the environment) and again 'deflates' to its usual size (minimal functionality).

```
repeat
    running(a1), running(a2), …, running(ap);
    anytime ∀ ai {
            if ( (∃ek ∈ R(ek)) ∧ (R(ek) = R(ai)) )
            then {
                aj ← GENERATE (aj , ek );
                send(ai, aj, REQUEST (ek ));
                when INFORM (aj , ai , 'done' )
                then ( REMOVE( ai , aj ) ∨ SELFREMOVE(aj) ) }
               endif
    endanytime
forever
ai ∈ PA, i = 1, …, p; aj ∈ OA; ek ∈ E.
```

Fig. 1. C3A Architecture Lifecycle

3 AjTempura as a C3A Application

Application of C3A model for development of real software will be demonstrated in this section. E-Learning environments providing electronic educational services are becoming an integral part of modern education. In the Faculty of Mathematics and Informatics at the University of Plovdiv, an infrastructure is being developed, known as Distributed e-Learning Center (DeLC), as a response to the need for supporting learning using modern information and communication technologies [14,11,13]. The center aims to provide adaptive and personalized e-learning services and teaching content located on physically separated servers. DeLC is a dynamic network structure consisting of [12]:

- Nodes – these operate as repositories of services and content;
- Relations – these specify various kinds of dependencies arising during an education experience.

The nodes can operate independently or dynamically link to each other, making complex virtual structures, called educational clusters. In the current DeLC versions, two educational clusters are built. The first cluster called MyDeLC is used to organize and perform e-learning by allowing fixed access to the services and electronic content via a specialized educational portal [15,16]. The second cluster, called InfoStation cluster offers a three-layer architecture which allows mobile access to services and informational resources via intelligent wireless access points (called Information Stations - ISs), situated around the university building [17,7,6].

In 2013 year we have started a new project that aims at the transformation of DeLC in a new infrastructure called Virtual eLearning Space (VeLSpace). In this space, the context-aware provision education services and teaching content will be supported by autonomous intelligent agents with 'limited rationality', i.e. changes can be identified only locally within the ranges of the agents positioned there. Each event causes a change in the local state of the range. In this way the state of the overall VeLSpace (global state) depends on the local states ordered in time. In order to manage the global state a suitable formalism, known as Interval Temporal Logic (ITL), was chosen as a theoretical model. Interval Temporal Logic [9] is a kind of temporal logic for describing time-dependent processes. Tempura interpreter [8] is an executable subset of ITL which uses the ITL syntax to identify time as a finite consequence of states. The existing version of Tempura was created in the C language [5] and repeatedly amended and expanded with new functionalities. Since the VeLSpace is built by separate intelligent autonomous components where the electronic services are equipped with their own operational agents, we created a new agent-oriented version of Tempura, called AjTempura, which can be easily integrated in VeLSpace while maintaining the space's homogeneity.

The transformation of the original Tempura interpreter was an iterative hand-made process which consist three basic steps. The software result of each step was tested and his run-time was compared with the original Tempura interpreter. Firstly we made a direct translation from C to Java code without changing the imperative structure of the interpreter. Second step was a refactoring process which aims at

creating an object-oriented version of Tempura called jTempura [19]. The final step was creating an agent-oriented version AjTempura. The AjTempura architecture is an implementation of the C3A model which is shortly described below.

The minimal functionality of AjTempura is implemented through two persistent agents - IOAgent and TempuraAgent. Both agents listen to the occurrence of events in VeLSpace. According to the kind of the identified event, an operative agent will be generated. This new agent has to analyze the event in more details and to initiate a corresponding compensatory action. AjTempura is implemented in development environment JADE [2]. The operating of the architecture is presented in the next diagrams.

Besides the two persistent agents, another one, known as Sniffer, is of key importance for tracking communication in agents. Its role is to provide a simple visual means for presenting the consecutive exchange of messages between the agents in a system. In its nature, the Sniffer agent draws diagrams similar to the Sequence diagrams in the UML language. The starting condition of this diagram, at the AjTempura start-up, can be seen in Fig. 2. In the left part of the figure, there is a list of the existing agents, among which there is a representative of the persistent IOAgent and a service DF agent. In accordance with AjTempura's lifecycle, the IOAgent starts sending periodic queries to DF for the presence of agents from the TempuraAgent type. The messages are sent every second until the reception of a response which contains a list of the present agents from the wanted type.

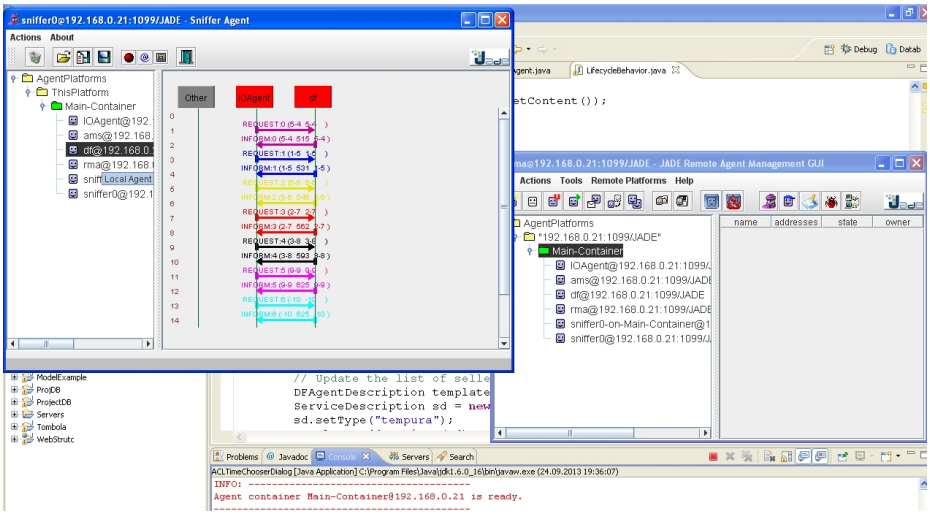

Fig. 2. Starting situation of the Sniffer agent

At the appearance of an agent from the TempuraAgent type in the DF list, an exchange of messages between this agent and IOAgent immediately takes place, which leads to the generation of an operative interpreting agent. The appearance of the necessary interpreting agent marks the start of the exchange of ITL sequences

between this agent and IOAgent until the IOAgent goals are met. The communication between the entire set of agents can be seen in the following state of the diagram, generated by the Sniffer agent (Fig. 3).

As a result from the appearance of the TempuraAgent and its communication with IOAgent, a new operative agent IA_1 has been generated. Its name is unique, and each following similar agent will be generated with a consecutive number. The operative agents exist until the IOAgent sends a 'done' message, i.e. confirmation for completion of the compensation action. In this case, this means that there are no more ITL sequences for processing and the IA_1 is redundant. After that the operative agents can be removed or self-removed (Fig. 4).

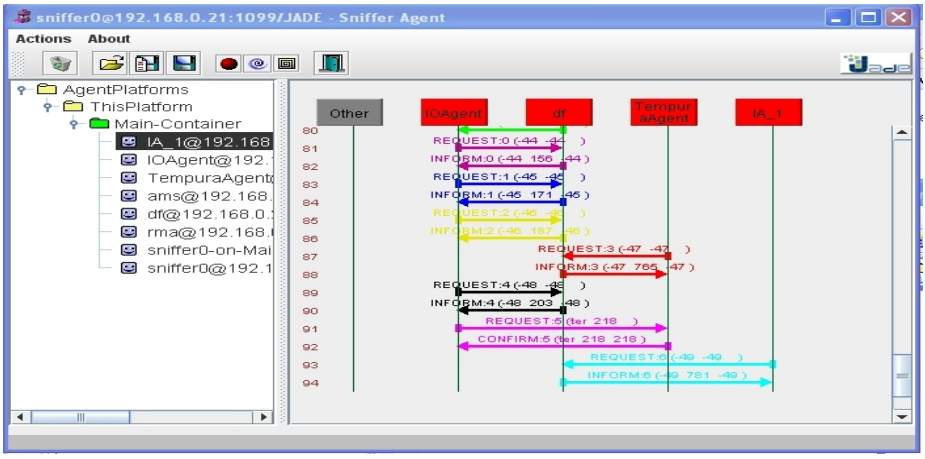

Fig. 3. Second view from the Sniffer agent

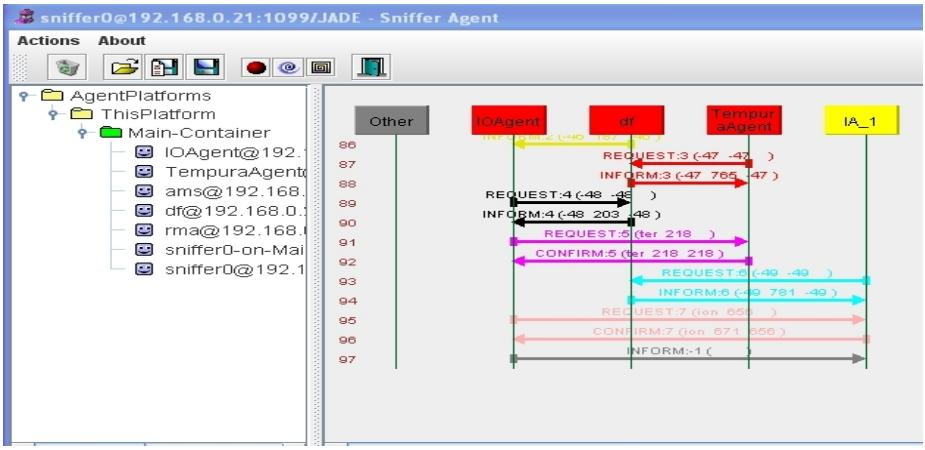

Fig. 4. Third view from the Sniffer agent

4 Conclusion

The current state of C3A model is presented in this paper. The research continues to expand and refine the existing version of the model. There are various issues being tackled. To name a few, for example, conflict resolving in overlapping or identical subspaces, saving traces of the removed operative agents as subintervals (the past), formalizing the interfaces between agents and education services, formalizing the notion of adaptation and personalization. The model extensions will be prototyped and examined in the VeLSpace.

Acknowledgment. This work is partially supported by FP7-REGPOT-2012-2013-1, grant agreement 316087 and Plovdiv University grant NI13-FMI-02.

References

1. Ashton, K.: That 'Internet of Things' Thing, in the real world, things matter more than ideas. RFID Journal (June 22, 2009)
2. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)
3. Dey, A.K.: Understanding and Using Context. Personal and Ubiquitous Journal 5(1), 4–7 (2001)
4. Abowd, G.D., Dey, A.K.: Towards a better understanding of context and context-awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999)
5. Hale, R.W.S.: Programming in Temporal Logic. PhD Thesis. Crambridge, England. Cambridge University (1988)
6. Ganchev, I., Stoyanov, S., O'Droma, M., Popchev, I.: Enhancement of International IEEE Conference on Intelligent Systems. In: 2nd International IEEE Conference on Intelligent Systems, Varna, pp. 359–364 (2004) ISBN: 0-7803-8278-1
7. Ganchev, I., Stoyanov, S., O'Droma, M., Popchev, I.: An InfoStation-Based University Campus System Supporting Intelligent Mobile Services. Journal of Computers 2(3), 21–33 (2007)
8. Moszkowski, B.: Executing Temporal Logic Programs. De Montford University, Cambridge (1985)
9. Moszkowski, B., Manna, Z.: Reasoning in interval temporal logic. In: Proceedings of the ACM/NSF/ONR Workshop on Logic of Programs, pp. 371–383 (1984)
10. Pascoe, J.: Adding generic contextual capabilities to wearable computers. In: 2nd International Symposium on Wearable Computers, pp. 92–99 (1998)
11. Stoyanov, S., Ganchev, I., Popchev, I., O'Droma, M.: An Approach for the Development of InfoStation-Based eLearning Architecture. Compt. Rend. Acad. Bulg. Sci., 62(9), 1189–1198 (2008)
12. Stoyanov, S., Ganchev, I., Popchev, I., O'Droma, M., Venkov, R.: DeLC -Distributed eLearning Center. In: 1st Balkan Conference in Informatics, Thessaloniki, Greece, pp. 327–336 (2003) ISBN: 960-287-045-1
13. Stoyanov, S., Popchev, I., Doychev, E., Mitev, D., Valkanov, V., Stoyanova-Doycheva, A., Valkanov, V., Mitev, I.: Educational portal. Cybernetics and Information Technologies (CIT) 10(3), 49–69 (2010)