tATAmI: A Platform for the Development and Deployment of Agent-based AmI Applications

Andrei Olaru, Marius-Tudor Benea, Amal El Fallah-Seghrouchni, Adina Magda Florea

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 476 – 483

Procedia Computer Science

The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

tATAmI: A Platform for the Development and Deployment of Agent-Based AmI Applications

Andrei Olaru[b], Marius-Tudor Benea[a,b,*], Amal El Fallah Seghrouchni[a], Adina Magda Florea[b]

[a]*LIP6 - University Pierre and Marie Curie, France*
[b]*Computer Science Department - University Politehnica of Bucharest, Romania*

Abstract

In the vision of a future pervaded by Ambient Intelligence (AmI), innovative solutions are required in order to facilitate the development of applications able to fulfill the real needs of the users. In using agents for building AmI applications, there is a lack of platforms and languages that strike a good balance between flexibility and power of expression, on the one hand, and ease of use and quick deployment, on the other hand. We introduce the tATAmI platform, which together with the S-CLAIM AOP language presents a suitable solution for these issues. This paper presents the architecture of tATAmI together with a brief description of two scenarios that were implemented using the platform and some other important technical aspects concerning it.

Peer-review under responsibility of the Conference Program Chairs
Keywords: ambient intelligence, multi-agent systems, agent-oriented programming, cross-platform deployment

1. Introduction

Ambient Intelligence (AmI) *provides a vision of the Information Society where the emphasis is on greater user-friendliness, more efficient services support, user-empowerment, and support for human interactions*[1]. After the mainframe and the personal computer, in the age of AmI the devices become invisible, being integrated in all objects surrounding the user and acting in concert for the fulfillment of his needs. The AmI applications are characterized, among others, by an intrinsic distribution of the architecture, by dynamics of the topologies and by frequent changes in the execution context. At a behavioral level, the need for context-sensitivity is a key element, allowing AmI applications to adapt to the various situations and users. Consequently, Multi-Agent Systems (MAS) and the Agent-Oriented Paradigm (AOP) have emerged as a well suited approach for the implementation of AmI applications[2,3].

Although there are many MAS development frameworks, like the ones briefly presented in Section 2, we still lack a good agent-based development framework for AmI that allows the development of applications to be deployed on a

* Corresponding author.
E-mail address: marius-tudor.benea@lip6.fr

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.018

hybrid network of devices (in which the mobile devices have an important role), that offers a good context representation and detection mechanism, while not hampering the autonomy of the implied agents, and which facilitates the interoperability with other platforms (sensors, cloud services, etc.). Furthermore, we would expect such a framework to be modular, for the needed and not-supported features to be easily integrated, to provide a deployment methodology that allows quick and repeatable experiments and to be easy to use, aiming to reduce the time to go from design of the agent system to its implementation and deployment. Last, the possibility to create powerful user interfaces should be offered.

Our solution to the above-mentioned problems is tATAmI[a] (towards Agent Technologies for Ambient Intelligence) an open-source platform for the development and deployment of agent-based AmI-applications that goes hand-in-hand with the S-CLAIM AOP language introduced in a previous work[4]. The applications developed using this framework can be deployed on networks of Java-enabled computers and Android mobile devices, with the help of the underlying Jade[b] framework. Moreover, a good context representation mechanism is provided, making use of hierarchical structures of agents together with the agent management and mobility capabilities offered by S-CLAIM, inspired from ambient calculus[5], and also making use of a flexible representation of agent knowledge together with a powerful knowledge matching mechanism. tATAmI is interoperable with web services and it can expose agents as web services as well. The definition of scenarios using XML, together with the use of a timeline for generating events and with visualization and simulation components facilitates the quick and repeatable execution of experiments. The use of the S-CLAIM language which is concerned only with the agent-related aspects of the applications, that is far less complex than general purpose languages and not as difficult to understand and learn, makes the framework easy to use. And finally, the complete separation of the GUI from the agent's code is a fertile ground for the development of powerful user interfaces.

In this paper we present the architecture of tATAmI together with a brief description of two scenarios that were implemented using the platform and some other important technical aspects concerning it. For more more information and for a more detailed explanation of the implementation of an example scenario, the readers could alse read the technical report available on the project's web page[c].

After presenting other relevant agent development frameworks in the next section, the paper details the structure of the platform and gives some insights on its implementation in Section 3. Deployment use cases are discussed in Section 4. The last section draws the conclusions.

2. Related work

The tATAmI platform has been created as a descendant of SyMPA[6], the platform destined to execute CLAIM, the language by which S-CLAIM has been inspired. However, tATAmI has been written from scratch. While there are several other MAS development frameworks, we believe that tATAmI strikes a good balance between power of expression and ease of use.

Repast Simphony[7] is a great environment for complex agent based modeling and simulations. This goal is, however, different from our main goal, the one of offering an agent based development environment for AmI, which supposed, in the first place, the possibility of deploying a MAS on a hybrid network of devices, with a very important accent on mobile devices. Agent Factory Micro Edition[8], instead, is a framework designed for AmI. It is a reduced footprint agent platform optimized constrained devices, namely devices running the CLDC / MIDP. They are, however, very limited limited and, with the development and spread of the modern mobile platforms and devices, they became outdated. For this reason we wanted tATAmI to offer the possibility of deploying cognitive agents on devices supporting Java and Android.

JIAC[9] is a Java based development framework and runtime environment for MAS, focused on industrial applications, in which the agents are integrated with the service-oriented architecture paradigm. However, it doesn't offer features needed by AmI applications, such as agent mobility and context representation. Moreover, JIAC does not offer an AOP development language. Jadex[10] is a BDI reasoning engine for programming goal-oriented agents in

[a] https://github.com/tATAmI-Project
[b] http://jade.tilab.com
[c] http://tatami-project.github.io/tATAmI-PC/files/tATAmI-report.pdf

Java and XML. Jadex can be used on top of different middleware infrastructures such as Jade and supports Android deployment. In many respects Jadex is similar to S-CLAIM and tATAmI. Jadex has, however, a more complex syntax and agent development requires more programming and theoretical knowledge. Moreover, AmI-specific requirements such as interoperability and context-awareness are not sufficiently satisfied. We find similar drawbacks in frameworks such as JACK Intelligent Agents[11], which brings the concept of intelligent agents into commercial software and Jason[12], a Java-based interpreter for an extended version of AgentSpeak.

JaCaMo[13], with its aim of covering all the aspects about programming autonomous agents, agent organizations and shared environments, by combining popular existing approaches for all three problems into a single, layered, framework is more appropriate for the development of AmI applications and has already been analyzed in this context[14]. A series of sub-projects related to it already offer support for the Android platform, for web services or for Arduino boards, which can make it a powerful tool in the field of AmI. However, it is still a work in progress and we also consider that separately programming all three layers mentioned is a heavy work that can generate many problems. That's why we consider a simpler, light-weight, approach as the one offered by S-CLAIM and tATAmI to programming MAS, while integrating most of the required features needed to program AmI applications.

3. Structure of the platform and implementation details

The tATAmI platform is structured on three main segments, or parts: the *Agent*, the *Simulation*, and the *Visualization*. Each agent in the deployment has the structure of the agent segment, which is also the *core* of tATAmI. The Simulation segment is the part that handles the deployment of agents and simulated events – we call *simulation* a complete execution of the agent system, from start to complete halt. The entire deployment is configured through an XML scenario file, so that no additional action from the user is required at execution. The Visualization segment provides the user with information on the agent system at a glance.

3.1. Agent

The Agent segment of the platform (see Fig. 1a) is an extension of, and relies on, the Jade agent. Therefore it inherits all the features offered by Jade: the possibility to run behaviors, the communication features and the mobility. The tATAmI agent features several additional aspects that are detailed in this section. An advanced developer is able to extend the agent to contain additional aspects, to which the existing features (parameters, logging, etc.) are also available.

The **Parametric aspect** assures the user of the platform that all that can be parameterized in the agent can be done so from the scenario file. Parameters set in the file are accessible to all aspects. For instance, a parameter may indicate a setting for the GUI, while another may be an initial parameter for the S-CLAIM aspect (such as the parameters `GUI` and `parent` in Fig. 3).

The **Visualizable aspect** of the agent covers several features related to easily tracing the agent's activity. It manages the agent's log, which prints its output in an area of the agent's GUI, and also sends the logging information to the central Visualization Agent, which gathers all logging information and sorts it according to timestamps. This aspect of the agent also automatically starts the appropriate, platform-specific GUI for the agent. The GUI may be the default one, or the developer may mention a specific GUI in the scenario file, depending on which the appropriate platform-specific implementation will be used.

The **Web Services aspect** offers two functionalities, implemented using Jade add-ons WSIG[15] and WSDC[16]. It allows the agent to access web services and use their output, in a way that is almost identical to communicating with other agents. Moreover, it exposes all of the S-CLAIM agent's behaviors as web services, so that they can be activated by, and receive messages from, other components and platforms that are not implemented using Jade.

The **Hierarchical aspect** enables the creation of logical hierarchies between agents, executing on the same machine or spread across several ones. These hierarchies are primarily used to create a partial representation of context[4] and to allow agents to communicate only within their context or to *hierarchically move* when their parent context moves.

Consider for instance the case of a user that works on a fixed workstation in a room in the building of a laboratory. The user's context is a hierarchical child to the room's context, which in turn is child of the building context. All contexts are represented by agents in the logical hierarchy. Say the user is using two context-specific services, also

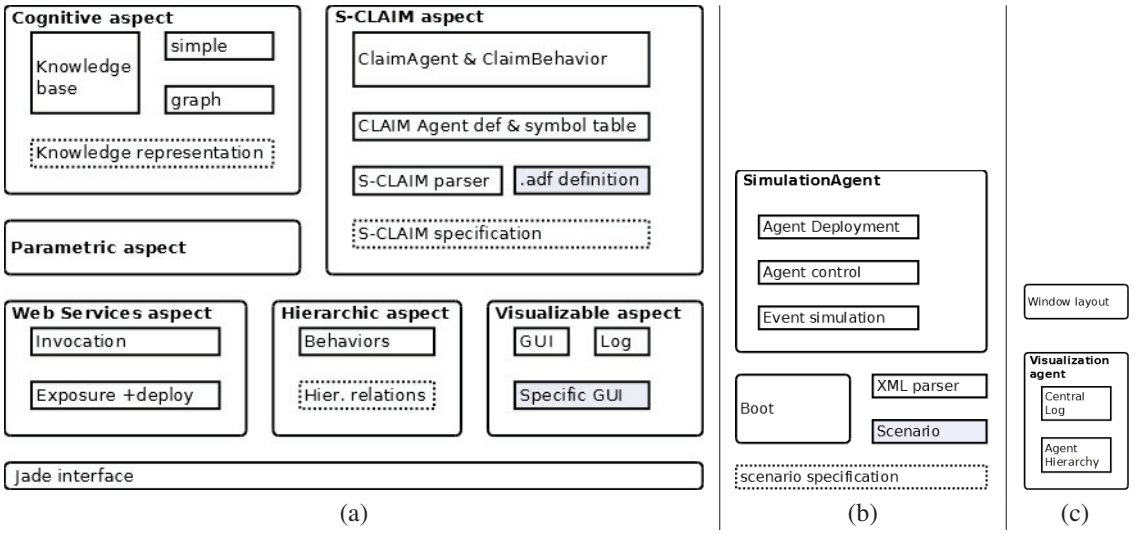


Fig. 1. The structure of the Agent (a), Simulation (b) and Visualization (c) segments. A dotted frame indicates specifications, not actually implemented elements. Highlighted frames indicate elements that need to be defined by the user for each specific scenario.

represented by agents, running on the workstation as hierarchical children of the user's personal agent: a navigation service that provides a map of the building, and a service that allows the user to control devices in the room. When the user moves to a workstation in a different room, its personal agent moves with it to the second workstation. The navigation service uses hierarchical movement to move as well. The other service, however, is specific to the context of the old room, and will therefore act as a *fixed* agent and remain on the original workstation.

The **Cognitive aspect** of the agent regards, for the time being, its use of a knowledge base. The type of the knowledge base need not be the same for all agents in a scenario, as any knowledge base implementation must be accessible through a standard interface offering three functionalities: adding knowledge, removing knowledge, and searching for knowledge that matches a certain pattern. Knowledge must be representable as an ASCII string and any aspect in the agent is able to access methods for adding, removing, reading and iterating knowledge items. See some examples in Fig. 4. More detailed examples are offered in a report published on the project's web page[c].

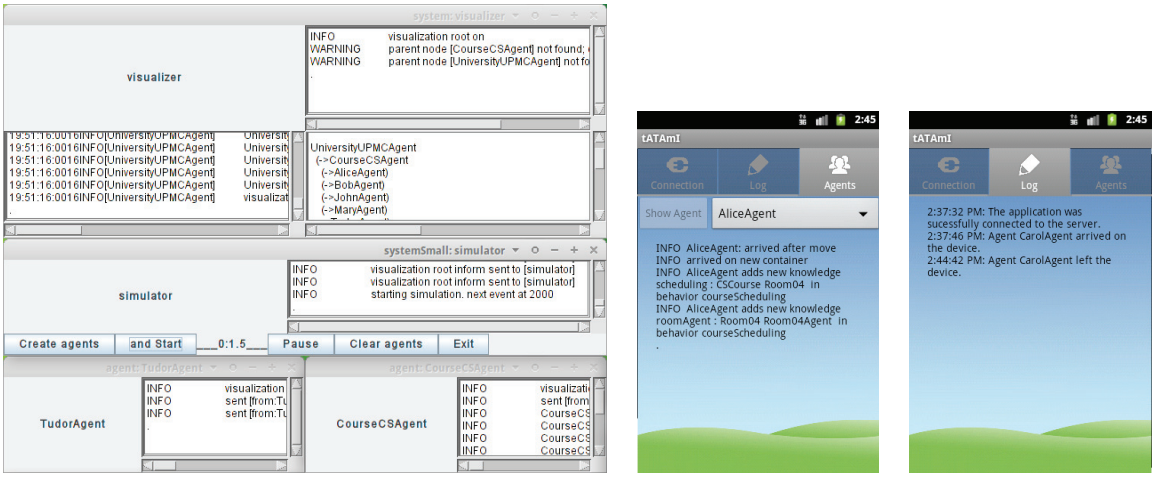
The **S-CLAIM aspect** is the top-most aspect of the tATAmI agent and uses all of the other aspects. An agent implemented using the S-CLAIM language[4] is based on an `.adf2` agent definition file (configured in the scenario file), that is parsed into an object containing multiple behavior definitions. Each one is transformed into a Jade behavior. Among the primitives there are communication primitives, primitives for knowledge management (`addK`, `removeK`, `readK`, `forAllK`), primitives for agent (hierarchical) mobility (`in`, `out`) and management (`new`, `open`, `acid`) and primitives that handle `input` and `output` to specific, name-designated components in the agent's GUI.

When developing S-CLAIM agents, there is a clear separation between agent design and algorithmic processing. The language is focused towards agent-specific operations (communication, mobility and knowledge management). Any algorithmic processing (e.g. working with numbers, processing input and output) is deferred to functions in Java libraries. Such functions can either be found in pre-existing packages, or are easy to implement using a minimum of procedural programming – they are pure functions, that take some input values, process them, and return an output.

3.2. Visualization and Simulation

The Simulation and Visualization segments of the platform (see Fig. 1b and 1c) are meant to allow the quick deployment and administration of agent-based applications.

A simulation is based on an XML scenario file. It contains all the information for configuring the Jade platform, for creating containers and agents, and on the parameters of each agent. It may also contain a timeline of "events", messages which are sent by the Simulation Agent to other agents in the scenario, to simulate external perceptions.



(a) (b) (c)

Fig. 2. Agent GUI on PC and Android platforms. On PC (a), agents feature separate windows, automatically laid out. Different agents have different sizes in the layout. On Android, the user can switch between the interface of each agent (b) or to the global log (c).

A snippet[d] of such a file is presented in Fig. 3. When starting the system, the scenario file is processed by a parser derived from SAX (Simple API for XML).

Once the scenario file is loaded and pre-processed, a *simulation* (a complete execution) can be started with two clicks – one to start the agents and one to initialize the events in the simulation.

Agents in tATAmI can be implemented in Java, but usually they are implemented using the advantages of S-CLAIM (as set by the `loader` parameter). Among the parameters of the agent in the scenario file is the `class` parameter, indicating the `.adf2` file to load for the agent. These agent definition files are loaded and parsed, using a LALR(1) parser generated using JFlex and BYACC/J, into complete descriptions of the agents and their behaviors. Each agent description is passed to a Jade agent, and each behavior description is passed to a behavior of the agent (behaviors can be initial, reactive or proactive[4]). For algorithmic processing, S-CLAIM code can use functions in Java libraries, which are called in exactly the same manner as built-in primitives. The implementation of such functions is easy: they are functions receiving a vector of arguments, some of which they can modify.

After agents are started, they are automatically deployed to the machines (containers) they are assigned to in the scenario, by using the mobility feature of Jade agents. On other machines, a stub scenario file is necessary, which specifies the name of the container(s) on that machine. The **Simulation Agent** informs all agents to whom to send the logging messages (i.e. to the Visualization Agent). When exiting, the agent first informs all other agents that they should exit, assuring a clean shut down of the platform.

While working with a multi-agent system deployed across multiple machines, it is important to be able to visualize how the system works, from a central point. This is what the Visualization component does, together with the Visualizable aspect of the tATAmI agent. The **Visualization Agent** receives logging messages from all agents in the platform, sorts them according to their timestamp (it is required that all machines have almost synchronous clocks), and displays them in its window. This is essential for following the activity of the system from a single point. Information about the logical hierarchy of the agents in the system is also displayed.

The **Window Layout** class is a small utility class that, based on simple layout indications, gives to every new window a space on the screen so that it does not overlap other windows. Various agents may have windows with specific sizes, for instance making the GUI of the Visualization agent larger than that of other agents. This way, the simulation starts with a clean layout, with no need to reposition windows by hand.

[d] More examples are available at `https://github.com/tATAmI-Project/tATAmI-PC/tree/master/scenario`

```
<scen:jadeConfig mainContainerName="Administration" platformID="SmartRoom" />
<scen:adfPath>scenario/2013/SmartRoom-EMAS</scen:adfPath>
<scen:agentPackage>agent_packages.example.smartRoom</scen:agentPackage>

<scen:initial>
  <scen:container name="AliceContainer" create="false">
    <scen:agent>
      <scen:parameter name="loader" value="adf2" />
      <scen:parameter name="class" value="StudentAgent" />
      <scen:parameter name="name" value="AliceAgent" />
      <scen:parameter name="parent" value="MASCourseAgent" />
      <scen:parameter name="userName" value="Alice" />
      <scen:parameter name="fixed" value="true" />
      <scen:parameter name="GUI" value="UserAgentGUI" />
    </scen:agent>
  </scen:container>
</scen:initial>
<scen:timeline>
  <scen:event time="2000" >
    <scen:CLAIMMessage>
      <scen:to>SchedulerUPMCAgent</scen:to>
      <scen:protocol>newSchedule</scen:protocol>
      <scen:content>
          ( struct message newSchedule ( struct knowledge scheduledTo CSCourse Room04 ) )
      </scen:content>
    </scen:CLAIMMessage></scen:event>
</scen:timeline>
```

Fig. 3. Declaration of AliceAgent in the scenario file, specifying the Jade configuration, locations agent files and functionality, and event timeline.

```
(agent Course ?courseName
  (behavior /*...*/
    (reactive changeRoom
      (receive (message scheduling ?courseName ?roomName))
      (addK (struct knowledge scheduling ?courseName ?roomName))
      (if (readK (struct knowledge roomAgent ?roomName ?roomAgentName)) then
        (forAllK (struct knowledge userAgent ??userName ??userAgentName)
            (send ??userAgentName (struct message scheduling ?courseName ?roomName ?roomAgentName))
        )
        (in ?roomAgentName)
      else (send parent (struct message whoManagesRoom this ?roomName))
      ) ) /*...*/  ))
```

Fig. 4. Part of the *CourseAgent* class, presenting an S-CLAIM behavior definition that receives a message with an updated schedule, sends the update to all children, and moves the agent as a child of the agent managing the new location.

3.3. Making tATAmI cross-platform

A tATAmI agent interacts with various features offered by the platform, such as agent GUI, web services, logging tools, and Jade itself, but must be deployed on various platforms, such as Windows, Linux, and Android. This portability was one of the reasons behind using Java for the implementation, more precisely JavaSE 1.6, as required by the Dalvik virtual machine[17].

In order to achieve this, several Java interfaces are used to abstract elements that are implemented differently on each platform. While the agent core uses exactly the same code on all platforms, there are different implementations for the relation with Jade, for visual interfaces, and for the logging infrastructure (Apache Log4J[18] is used on PC platforms and the Java logger is used on Android). Exposing agents as web services is a feature not available while the agent is on an Android device. Execution on Android is done by means of the *JadeLeapAndroid* add-on for Jade.

Each time the agent moves, it destroys its visual presence when leaving a machine and it recreates its GUI on the machine it moves to, creating a GUI that depends on the platform and on the settings in the scenario file. Interfacing with the GUI is done through a set of GUI *components*, identified by names that are common between the S-CLAIM code and the GUI implementations on both PC and Android. In the S-CLAIM code, the developer specifies that input should come from, or output should go to, a component with a specific name (the `input` and `output` primitives are used). Inputs come in two flavors: *passive* inputs are meant to have their value read on request; *active* inputs can also activate behaviors when used by the user (e.g. a button).

4. Deployment use cases

The tATAmI platform has already been successfully used in several projects. The **SmartRoom scenario**[4], developed together with the WSN team at Honiden Lab in the National Institute of Informatics in Tokyo[e] is one of them. In the scenario, there are three students enrolled in the MAS course at the UPMC University. The students are Alice, Bob and Carol and each of them has his/her own smartphone, running the SmartRoom application.

There are agents for each context in the scenario: the `UniversityUPMCAgent` and `SchedulerUPMCAgent` manage the spacial and temporal contexts of the university. The `Room04Agent` manages the context of a room on the campus. And the `MASCourseAgent` (of the type `Course` – see Fig. 4) manages the context of the course activity. Each student has an associated agent, managing user context, running on his/her behalf (`AliceAgent`, `BobAgent` and `CarolAgent`, of the type `StudentAgent`).

In the beginning, the agents of the students are children (in the logical hierarchy) of the Course agent, as the students are currently part of that activity. When the Scheduler agent announces the Course agent that the course has been moved to Room 04, the Course agent moves to the container where `Room04Agent` is located (the activity will take part in that particular spatial context). The student agents don't move, as they are fixed to their devices.

Should the students take part in course sub-activities that imply more interaction, their agents may move from their devices to larger screens where students may work together. We have successfully demonstrated that in the SmartRoom, where the student's agents moved from one machine / screen to another as the Smart Room detected the movement of the student from one activity group to another[4]. The location services where offered by means of a web service interface, which tATAmI operated with seamlessly, just as with any other agent in the system.

The **Pro-Con debate application**[f] was a successful diploma project with students at University Politehnica of Bucharest, that involved the development of an AmI application using S-CLAIM and tATAmI, also deployed on multiple PC's and Android devices. In a similar fashion with the SmartRoom scenario, there are several workstations with large screens and several Android devices belonging to students. On each workstation there is an agent managing a discussion group ('Pro' or 'Con'). Students add opinions, using their smartphones, to the set of opinions of the group. After a while, groups exchange students. When a student moves to the other group, his/her opinions are removed from the screen he/she leaves and appear on the screen he/she moves to. This happens automatically, through agent mobility between machines and through communication between the student's agents and agents managing their current context.

While implementing the scenarios above, the ease of developing agents in S-CLAIM and deploying them in tATAmI became apparent. The agent developer must only define behaviors for each type of agent, and describe each behavior in terms of communication, mobility, and knowledge management primitives. All the parameters for the simulation are given in the XML file, which has a clearly defined schema and intuitive names for the parameters of agents. The progress of the execution of the agents is easy to follow using the centralized or agent-local logs. In the case of the SmartRoom collaboration, the team in Tokyo successfully deployed the platform following only minimal instructions and little communication between LIP6 and NII.

5. Conclusion and perspectives

In this paper we have introduced tATAmI, a platform for the easy and quick implementation of mobile MAS, interoperable with web services and that can be deployed on heterogeneous networks of Java and Android devices. Together with the advantages offered by the light-weight AOP language S-CLAIM, and using features such as pattern-based knowledge matching and hierarchical mobility, tATAmI represents a viable alternative for the development of agent-based AmI applications.

We have detailed the architecture and the implementation of the platform, proving how tATAmI fulfills some essential requirements in the development of AmI applications: developing agents is easy, focused on agent-specific issues, and deploying them requires just a few clicks; experiments are easy to perform and to repeat, thanks to scenario

[e] `http://www.honiden.nii.ac.jp/en/research/wireless-sensor-network`

[f] We thank Iulia Moscalenco and Miruna Popescu for development of the application for their diploma projects.

files, to the automatic deployment of agents on multiple machines, and to a visual environment that is easy to keep track of; agent-orientation makes the execution units of AmI applications act autonomously, be oriented on behaviors, and be mobile across the system, also taking advantage of a context-based logical hierarchy that facilitates hierarchical communication and movement. The tATAmI platform has been used in several occasions, such as the ones mentioned in Section 4, for the deployment of agent-based AmI applications, and positive feedback has been obtained.

Some of our priorities for the period to come are to increase the usability of the platform by providing a better documentation and maintenance. We also desire to improve web services support for Android devices. We wish, as well, to focus on the creation of a library of commonly used functions to be easily accessed and used by agent developers. The goal-oriented aspect of agents will be further explored.

Acknowledgements

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreements POSDRU/159/1.5/S/132395 and POSDRU/159/1.5/S/134398.

References

1. K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, J. Burgelman, Scenarios for ambient intelligence in 2010, Office for official publications of the European Communities, 2001.
2. F. Sadri, Ambient intelligence: A survey, ACM Comput. Surv. 43 (4) (2011) 36:1–36:66. doi:10.1145/1978802.1978815.
3. C. Ramos, J. C. Augusto, D. Shapiro, Ambient intelligence - the next step for artificial intelligence, IEEE Intelligent Systems 23 (2) (2008) 15–18.
4. V. Baljak, M. T. Benea, A. E. F. Seghrouchni, C. Herpson, S. Honiden, T. T. N. Nguyen, A. Olaru, R. Shimizu, K. Tei, S. Toriumi, S-CLAIM: An agent-based programming language for AmI, a smart-room case study, Procedia Computer Science 10 (0) (2012) 30 – 37, {ANT} 2012 and MobiWIS 2012. doi:10.1016/j.procs.2012.06.008.
5. L. Cardelli, A. D. Gordon, Mobile ambients, Theor. Comput. Sci. 240 (1) (2000) 177–213.
6. A. Suna, A. Fallah-Seghrouchni, A mobile agents platform: Architecture, mobility and security elements, in: R. Bordini, M. Dastani, J. Dix, A. Fallah Seghrouchni (Eds.), Programming Multi-Agent Systems, Vol. 3346 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 126–146. doi:10.1007/978-3-540-32260-3_7.
7. M. North, N. Collier, J. Ozik, E. Tatara, C. Macal, M. Bragen, P. Sydelko, Complex adaptive systems modeling with Repast Simphony, Complex Adaptive Systems Modeling 1 (1). doi:10.1186/2194-3206-1-3.
8. C. Muldoon, G. O'Hare, R. Collier, M. O'Grady, Agent Factory Micro Edition: A framework for ambient applications, in: V. Alexandrov, G. van Albada, P. Sloot, J. Dongarra (Eds.), Computational Science ICCS 2006, Vol. 3993 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 727–734. doi:10.1007/11758532_95.
9. M. Lützenberger, T. Küster, T. Konnerth, A. Thiele, N. Masuch, A. Hessler, J. Keiser, M. Burkhardt, S. Kaiser, S. Albayrak, JIAC V: A MAS framework for industrial applications, in: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013, pp. 1189–1190.
10. A. Pokahr, L. Braubach, W. Lamersdorf, Jadex: A BDI reasoning engine, in: R. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (Eds.), Multi-Agent Programming, Vol. 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer US, 2005, pp. 149–174. doi:10.1007/0-387-26350-0_6.
11. N. Howden, R. Rönnquist, A. Hodgson, A. Lucas, JACK Intelligent Agents-summary of an agent infrastructure, in: 5th International conference on autonomous agents, 2001.
12. R. Bordini, J. Hbner, BDI agent programming in AgentSpeak using Jason, in: F. Toni, P. Torroni (Eds.), Computational Logic in Multi-Agent Systems, Vol. 3900 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 143–164. doi:10.1007/11750734_9.
13. O. Boissier, R. H. Bordini, J. F. Hbner, A. Ricci, A. Santi, Multi-agent oriented programming with JaCaMo, Science of Computer Programming 78 (6) (2013) 747 – 761. doi:http://dx.doi.org/10.1016/j.scico.2011.10.004.
14. A. Sorici, O. Boissier, G. Picard, A. Santi, Exploiting the JaCaMo framework for realising an adaptive room governance application, in: Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, SPLASH '11 Workshops, ACM, New York, NY, USA, 2011, pp. 239–242. doi:10.1145/2095050.2095088.
15. D. Greenwood, JADE web service integration gateway (WSIG), Whitestein Technologies, Jade Tutorial, AAMAS'05.
16. E. Scagliotti, G. Caire, Web services dynamic client guide (2009).
17. D. Ehringer, The Dalvik virtual machine architecture, Techn. report (March 2010).
18. S. Gupta, Pro Apache Log4j, Vol. 2, Springer, 2005.