Virtual Agent Modeling in the RASCALLI Platform

(Special Session on EU-projects)

Christian Eis
Research Studios Austria
Vienna, Austria
christian.eis@researchstudio.at

Marcin Skowron
Austrian Research Institute for Artificial Intelligence
Vienna, Austria
marcin.skowron@ofai.at

Brigitte Krenn
Austrian Research Institute for Artificial Intelligence,
Research Studios Austria
Vienna, Austria
brigitte.krenn@ofai.at

ABSTRACT

The RASCALLI platform is both a runtime and a development environment for virtual systems augmented with cognition. It provides a framework for the implementation and execution of modular software agents. Due to the underlying software architecture and the modularity of the agents, it allows the parallel execution and evaluation of multiple agents. These agents might be all of the same kind or of vastly different kinds or they might differ only in specific (cognitive) aspects, so that the performance of these aspects can be effectively compared and evaluated.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents; D.2.11 [**Software Architectures**]: Domain-specific architectures; K.6.3 [**Software Management**]: Software development

General Terms

Design, Experimentation, Measurement

Keywords

Cognitive agents, Agent modeling, Agent evaluation, Open source software

1. INTRODUCTION

This paper gives an overview of the architecture and functionality of the RASCALLI platform, developed as part of the RASCALLI project[1]. In this project, the platform is used as the underlying software environment for the development and execution of so called RASCALLI (Responsive Artificial Situated Cognitive Agents that Live and Learn on the Internet). It provides the facilities for user-agent and agent-agent communication and serves as a testbed for the evaluation of various incarnations of the agents that use different sets of action-perception tools, action selection mechanisms and knowledge resources. The platform supports a modular development style, where agents are assembled from small re-usable building blocks. Agents of different kinds and configurations can run simultaneously within a single platform environment. This enables the evaluation and comparison of different agents as well as the evaluation of whole agent communities.

The paper is organized as follows: Section 2 describes the general objectives of the RASCALLI project to the extent that they are relevant for the design and implementation of the RASCALLI platform. Section 3 gives a brief overview of related platforms and methodologies. The RASCALLI platform itself is then described in section 4, and section 5 gives an overview of the agent components implemented in the RASCALLI project. Finally, section 6 explains how the RASCALLI platform can be used for agent evaluation.

[1]European Commission Cognitive Systems Project FP6-IST-027596-2004 RASCALLI.

http://doi.acm.org/10.1145/nnnnnn.nnnnnn

2. PROJECT OBJECTIVES

The project RASCALLI aims at the development of virtual agents that perform tasks related to accessing and processing information from the Internet and domain-specific knowledge bases. RASCALLI agents, further referred to as Rascalli, represent a growing class of cooperative agents that do not have a physical presence, but nevertheless are equipped with major ingredients of cognition including situated correlates of physical embodiment to become adaptive, cooperative and self improving in a virtual environment, given certain tasks.

The project objectives cover the following topics: development of a computational framework for the realization of cognitive agents providing intelligent assistance capabilities; cognitive architecture and modeling; perception and action; reasoning; learning; communication; agent-to-agent and agent-to-user interfaces. The Rascalli answer questions of their users, learn the users' preferences and interests, and use this knowledge to present the users with new, interesting information. The agents exist in an environment consisting of external knowledge sources on the Internet, such as search engines and RSS feeds, and domain-specific knowledge bases. They communicate with their users as well as with other Rascalli.

Rascalli are developed in a modular fashion, which allows individual agents to be built from different sets of components. This enables e.g. the creation of agents which are "ex-

perts" in different knowledge domains. During its life-time, each agent adapts to its user's interests and thus further specializes in a certain sub-domain and evolves in accordance with the user's preferences. This results in a community of specialized agents, which can communicate with each other to provide requested information to their users.

The modular approach enables the evaluation and comparison of different sets of components, including different cognitive aspects (e.g. different learning strategies) by executing multiple Rascalli in the same environment and evaluating their performance (e.g. by means of user tests).

For the realization of these objectives, system integration turned out to be a major roadblock, due to the following reasons:

- Even though Java was chosen as the main implementation language for the project, some project partners have no or little experience with Java development.
- In order to avoid re-implementation, we had to integrate existing components from previous projects. These components are based on a wide range of technologies, such as different programming languages (e.g. Perl, Java, Lisp) and even native binaries for different operating sytems (Linux and Windows).
- Initial attempts at providing a development environment that integrates all of these components and can be replicated on each developer's machine proved to be difficult to use and keep up-to-date.

Based on the project objectives and constraints outlined above, we arrived at the following **set of requirements** for our software platform:

- Support the execution of various agents, belonging to different users.
- Support agent-to-agent and agent-to-user communication.
- Allow developers to implement diverse agents based on shared components (this also means that multiple versions of each component can exist at the same time).
- Integrate external and legacy components with minimal effort.
- Build agents in a modular, component-based fashion.
- Build the platform itself in a component-based, extensible fashion.

3. RELATED WORK

Research into related work has been conducted in multiple directions, including multi-agent platforms, as well as platforms and development methodologies for cognitive systems. While many such platforms and methodologies exist, none of them meets the requirements set for the RASCALLI platform.

3.1 Multi-Agent Platforms

We have investigated FIPA[2] compliant agent platforms, such as JADE[3] [1], which is a Java-based middleware for multi-agent systems. However, these systems mostly focus on distributed systems and communication issues. Also, the RASCALLI platform is not a multi-agent system in the traditional sense, where agents are independent components of a larger application, working for a common goal. Instead, Rascalli are complete individual entities that simply happen to share the same environment and may communicate with each other, if they wish. Furthermore, none of the investigated agent platforms supports the development style targeted by the RASCALLI platform, where multiple agent architectures and agent definitions, as well as multiple versions of agent components co-exist in a single platform instance.

It might be interesting future work to implement or integrate some of the FIPA standards (such as the Agent Communication Language) with the RASCALLI platform.

3.2 Agent Development Methodologies and Frameworks

As an example of an agent development methodology, Behavior Oriented Design (BOD)[4] [2] supports the implementation of agents based on an iterative development process and a modular design. However, it does not provide much of a runtime environment. Therefore, BOD does not really compare to the RASCALLI platform, even though both advocate a modular approach. It would, however, be interesting to implement an agent architecture based on BOD within the RASCALLI platform and thus use the platform as a runtime environment for BOD agents.

As for development frameworks, AKIRA[5] [7] aims to create a C++ development framework to build cognitive architectures and complex artificial intelligent agents. However, like the FIPA multi-agent platforms, it targets different requirements than the RASCALLI platform. It might be interesting to consider exploiting some of AKIRA's concepts within the RASCALLI platform.

4. RASCALLI PLATFORM

This section provides an overview of the RASCALLI platform. We start with a set of features supported by the platform in order to fulfill the requirements listed above. Then we describe the software architecture of the platform and explain how this architecture supports the various platform features.

4.1 Platform Features

Multi-Agent: The platform supports the concurrent execution of multiple agents, including agents of the same kind, as well as agent of different kinds, ranging from very similar to vastly different.

Multi-User: Each agent has a single user (but one user may have several agents).

[2]http://www.fipa.org/
[3]http://jade.tilab.com/
[4]http://www.cs.bath.ac.uk/ai/AmonI-sw.html
[5]https://sourceforge.net/projects/a-k-i-r-a/

Agent Layer	Agent architectures, components, definitions and instances
Framework Layer	Technical services and utilities (e.g. networking support, RDF support, logging support)
Infrastructure Layer	Basic tools and components (e.g. Java, OSGi, Maven)

Figure 1: Platform Layers

Communication: Agent-to-agent communication can be implemented on the Java level, since all agents run within a single runtime environment. Alternatively, agents can communicate via instant messaging (this opens the possibility to use multiple, distributed platform instances). Several channels for agent-to-user communication have been implemented (currently a proprietary protocol for the 3D client interface, Jabber instant messaging and email).

Component-Based Architecture: All the parts that are required to set up a specific agent are developed in a component-based fashion so that individual Rascalli can be assembled from these components in a Lego-like manner.

Extensibility: The platform itself is built in a component-based fashion and can therefore be easily extended, even at runtime.

Distributed, Concurrent Development: A single platform instance is shared by all developers for implementing different agents, with minimal interference between individual developers.

Multi-Version: In order to support the concurrent development of different kinds of agents based on shared components, the platform supports the execution of multiple versions of the same components at the same time.

System Integration: External (and possibly legacy) components need to be integrated only once and are then available to all agents running in the platform in an easy-to-use manner. Since only one instance of the platform is required, there is no need to duplicate the entire software environment on multiple developer machines.

4.2 Platform Architecture

The RASCALLI platform is implemented as a layered architecture (Fig. 1).

4.2.1 Infrastructure Layer

The Infrastructure Layer contains basic tools and components used in the RASCALLI project. Specifically, these are Java, Maven[6] and OSGi[7]. In addition, this layer contains custom-made development and administration tools for the RASCALLI platform, such as user interfaces for agent configuration and deployment tools.

The most important feature of this layer is the use of OSGi, which implements a dynamic component model on top of Java. This means that components can be installed, started, stopped and uninstalled at runtime. Furthermore, dependencies between components are managed by OSGi in a fashion that allows the execution of multiple versions of a single component at the same time. Finally, OSGi provides a framework for service-based architectures, where each component can provide services to other components, based on Java interface specifications.

[6]http://maven.apache.org/
[7]http://www.osgi.org/

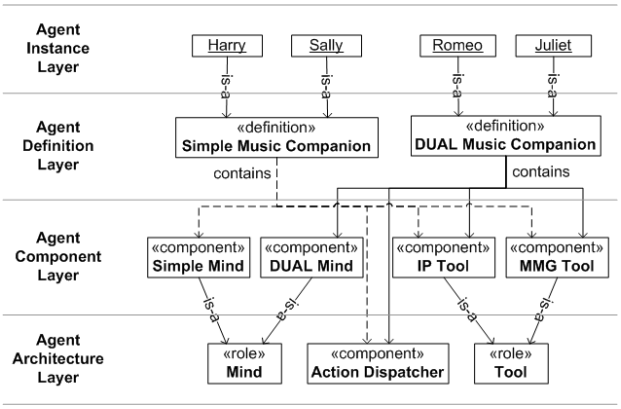

Figure 2: Agent Layer

The use of OSGi thus enables the platform features 'multi-version' and 'extensibility', and supports the implementation of a 'component-based architecture' in the upper two platform layers.

'Distributed, concurrent development' is enabled by Maven and some custom-made components on this layer.

4.2.2 Framework Layer

The Framework Layer comprises general platform services and utilities employed by the Rascalli, including communication (user to agent, agent to agent), event handling, RDF handling, technology integration (Perl, web services, etc.), and various other platform services.

The services on this layer implement the 'multi-agent', 'multi-user' and 'communication' features of the platform. Furthermore, this is the place where 'system integration' takes place. External components are integrated and made available to the components of the Agent Layer as OSGi services, which can then be accessed on the Java level.

4.2.3 Agent Layer

The Agent Layer is the application layer of the platform and contains the implementation of the actual agents. It is designed to support the development and execution of multiple agents of different kinds as required by the project objectives. This layer consists of the following sub-layers:

Agent Architecture Layer: An agent architecture is a blueprint defining the architectural core of a particular type of Rascalli. More precisely, it sets the roles of agent components and provides means for defining and assembling a specific agent. The architecture can also contain implementations of common components shared by all agent definitions.

Agent Component Layer: Contains implementations of the roles defined on the Agent Architecture Layer.

Agent Definition Layer: An agent definition is an assembly of specific components of the Agent Component

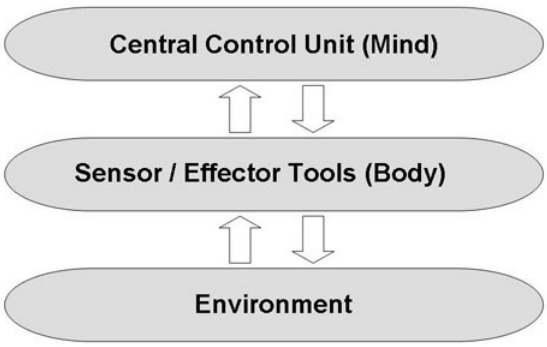

Figure 3: MBE agent architecture

Layer of a specific agent architecture. Different agent definitions for the same agent architecture might contain different components for certain roles.

Agent Instance Layer: Contains the individual agent instances. Each Rascallo is an instantiation of a specific agent definition.

Fig. 2 gives a (simplified) example of an agent architecture (the Mind-Body-Environment architecture, see section 5). The Agent Architecture Layer defines two roles, Mind and Tool, and implements an agent component (Action Dispatcher) shared by all agent definitions. The Agent Component Layer contains two implementations of the Mind role, as well as two Tools. Based on this architecture, two agent definitions combine each of the Mind implementations with the available Tools and the Action Dispatcher into different kinds of agents. Finally, a number of agent instances are shown on the Agent Instance Layer.

5. AGENT COMPONENTS

This section gives an overview of the agent components which have been implemented in the RASCALLI project (for more detailed information see [10]). RASCALLI agents are currently based on a reactive agent architecture with a perception-action loop. This agent architecture is called Mind-Body-Environment architecture, because each agent has a central control unit (Mind), which perceives and acts on the Environment via a set of sensor and effector tools (Body), as shown in Fig. 3.

The Environment comprises the agent's user, other agents, Internet services and domain-specific knowledge bases. The agent perceives its environment (via a set of perception sensors, implemented as software tools) as a set of virtual entities with their own characteristics and properties. These include strings of written language originating from the user or extracted from HTML documents, markup tags, information about the accessibility of various Internet and local tools and resources, user feedback, etc. Therefore, the agent has to deal with a dynamic environment i.e. evolving content of the websites, permanent or temporary inaccessibility of Internet services, appearance of new content or services, changes in the user preferences and interests, as well as the natural language input from the user and the web-pages. Similarly, all actions of an agent in its environment are performed on the above introduced set of entities.

The Body contains components called Tools, which serve as sensors (Perception Layer) and effectors (Actuator Layer). A specific agent definition may contain an arbitrary subset of the available Tools, but of course, the chosen Mind component must be able to deal with the selected Tools. Some of the available Tools are:

- **T-IP4Dual, T-IP4Simple and T-IP4Adaptive**, Input Processing Tools transform natural language and user feedback inputs into categorized information useable by the respective Mind components, and serve as a Perception Layer of an agent.

- **T-MMG**, for generating multi-modal output to the user. The Multi-Modal Generation Component provides a middle-ware functionality between generated agent output and the user interfaces. The generation component implements a template-based approach (in form of Velocity templates) by encoding vocabulary, phrases, gestures etc., which can be combined with the output of the RASCALLI Tools and context data. The use of Velocity[8], a template generation engine, allows templates to be designed and refined separately from the application code.

- **T-QA**, a general purpose open-domain question answering system (based on the work described in [8], [9]) is used in the RASCALLI platform to provide answers to the user factoid-type questions expressed in natural language.

- **T-Nalqi**, a natural language database query interface. The Tool is used in the RASCALLI platform for querying the databases accessible to the Rascalli, in a search for instances and concepts that can provide answers to the user questions. The component analyses natural language questions posed by the user and retrieves answers from the system's domain-specific databases.

- **T-RSS**, provides a mechanism for Rascalli to retrieve current information that might be of interest for the user.

- **T-Wikipedia**, an interface and analysis Tool for Wikipedia.

The Mind component performs action selection, based on the current input from the environment and the agent internal state. It can also make use of supporting services, for example a user/agent modeling service. The following Mind components are being developed in the RASCALLI project:

- **Simple Mind**, which performs action selection with a simple rule-based mechanism. These rules match to specific cues in the input data arriving from sensor channels. 'Simple Mind' extracts relevant information and passes this information on to the appropriate effector tool. Even though seemingly non-trivial behavior can be accomplished through a series of interactions of the Simple Mind and the available tools, the Simple Mind does not contain any cognitive aspects such as memory or learning.

[8]http://velocity.apache.org

- **DUAL Mind**, which incorporates the DUAL/AMBR ([5], [6]) cognitive architecture for action selection. It includes a long term memory (LTM) where general and episodic knowledge is stored, and a working memory (WM) constituted by the active part of LTM, perceptual input and goals. The DUAL mind operates only on represented knowledge and has only a mediated connection to the body and the environment. Thus it contains a partial, selected representation of the environment at an abstract conceptual level and experiential memories related to specific episodes like organization of the interaction of an agent with its environment.

- **Adaptive Mind**, a machine learning based classification driven action selection mechanism. Based on the available knowledge, including the perception of an input situation, the agent finds a set of actions that can be applied to a given input situation. The selection of a particular action is based on the similarities with other actions the agent had successfully performed in the past, i.e. received positive feedback from the user. The action selection classifiers are implemented as Maximum Entropy models [4]. The data used for training the classifiers represents an input situation in terms of entities from the environment perceived by the agent, an action applied to this situation and the feedback obtained from the user.

RASCALLI agents come with a variety of user interfaces comprising a 3D client featuring an embodied conversational character [3] (ECA-UI, see Fig. 4 and 5), a Jabber instant messaging integration, a web-based user interface (Web-UI) and two domain-specific, special purpose interfaces which allow the user the explore in a playful way the domain-specific knowledgebases accessible to the currently implemented Rascalli. The Web-UI[9] mainly serves for user registration, download of the 3D-client and specification of Internet resources (URLs and RSS-feeds) that are considered by the user as important to be monitored by the agent. The agents use the Web-UI to present list-like information to the user, and more generally all information not well suited for presentation by means of synthesized speech. The ECA-UI, on the contrary, specializes on virtual human-to-human dialogue. In order to avoid the bottelneck imposed by speech recognition, user input is restrained to utterances typed into a small text window and to pressing buttons in order to praise or scold the agent. The user is expected to ask domain-specific questions but may also engage in a chatterbot-type of conversation with the ECA. To do this the Rascalli make use of ALICE chat bot technology.[10]

The knowledge sources are a music database featuring songs and albums of more than 60,000 singers/musicians and a database providing background information on musical artists such as their family relations, religion, track record, band membership etc.[11]

Based on these components, multiple agent definitions have been conceptualized and developed, including the initial implementation of an agent with a basic set of tools (sandbox for testing the system components and their integration), an agent utilizing an implementation of the DUAL cognitive architecture as central control unit, and a Smart Music Companion. Using those various incarnations of the agents implemented in the RASCALLI platform the performance (in terms of user satisfaction) can be compared and evaluated.

[9]http://intralife.researchstudio.at/rascalli/
[10]http://www.alicebot.org/downloads/programs.html
[11]See http://rascalli.researchstudio.at/ and http://rascalli.dfki.de/ontology/ for browsing the data.

6. AGENT EVALUATION

In the following, we first give an overview of evaluation scenarios supported by the RASCALLI platform in general and the currently implemented Rascalli in particular. We then give a more detailed account of an evaluation scenario the goal of which is to investigate how the use of cognitive aspects in an agent can improve the user experience.

6.1 Evaluation Scenarios

We distinguish three kinds of scenarios:

1. Automated performance measures: The high modularity of the platform eases the integration of automated performance measures with existing and future agent architectures. For example, one could easily measure the time an agent needs to fulfill a given task, as well as the accuracy with which an agent performs certain tasks.

2. Data mining from user activity logs: All user activities are logged in the platform including user id, agent id, timestamp and activity type. These data can then be evaluated employing data mining techniques and other quantitative evaluation methods, in order to identify prevalent usage patterns of individual users and across different users, as well as different kinds of agents.

3. User testing: We distinguish two kinds of user testings. The one are studies where users interact with an agent for a short time to fulfill a certain, narrowly defined task. The other one are studies where users interact with their agents more freely for a longer period of time. While with the former the focus lies on testing specific aspects of the system and the related human-computer interaction, the latter address more general questions about what makes an agent a suitable companion for its user, which interfaces support which tasks and how the interaction of the user with the agent and his/her attitude towards, his/her liking and understanding of the companion changes over time.

As there are currently no automated performance measures built into the RASCALLI platform, and the platform has not yet been made accessible to a broader public and thus we still lack suffient amounts of usage data for applying data mining techniques, we will concentrate in the following on user testing. Due to its modularity, the platform allows us to experiment with agents being placed in the same environment, but assembled from different internal building blocks and therefore are equipped with different perception and action capabilities as well as decision making strategies.

A male (Fig. 4) and a female (Fig. 5), human-like version of the 3D character have been implemented. Both characters are built on the basis of the same body model and are equipped with a similar set of gestures and facial expressions.

Figure 4: ECA user interface (male)

Figure 5: ECA user interface (female)

They are also comparable as regards their appearance, both are designed to fit a pop-rock scenario. The identity of the characters except for gender related optical features allows us to study effects that may be attributed to different perception of gender by simply switching between the male and female character whereas all other parameters in the system are kept the same. The RASCALLI system enables us, without further programming, to experiment with all kinds of settings where we let a user interact with or train his/her agent making use of either the male or the female version of the 3D character and then switch to the opposite sex.

The availability of two conceptually different, complementary UIs for user-agent communication, i.e., the Web-UI for standard windows and menu-based human-computer interaction and the ECA-UI for interaction closely related to human multimodal dialogue, is a valuable prerequisite to investigate user preferences for different modes of interaction and social implications of human-computer interfaces.

At the time of writing two user studies are completed: the one (S1) concerning the liking and evaluation of individual users engaging in a prescripted question-answer communication with the male Rascalli character, the other one (S2) is a study concerning the usability of the interface to the music data. Two further user studies are under preparation. The one (S3) is a replication of the question-answer study employing the female character instead of the male one. Whereas the previously mentioned studies are all one-time encounters with the system, the second one (S4) of the studies under preparation is a longer-term study where the users interact with and train their agents over a longer period of time, with user assessments at the beginning, at several intermediate stages and at the end of the agent-user collaboration period. Amongst others, we are interested in changes over time of the users' liking of their agents, their expectations on the performance of their agents, their accounts of trust, and their strategies to adapt to their agents in order to achieve/satisfy their information requirements in collaboration with their agents.

In particular, we prepare two variants (narrow and broad) of the longer-term study. In the narrow study (S4.1), the task is to train the agent as good as possible to monitor RSS-feeds for a specific topic of interest. To do so, the users are asked to identifiy and sharpen a topic of interest such as artist, group, genre, song, album etc. by exploring the music-related knowledge of their agent utilizing the agent's domain-specific interfaces. In addition, they are asked to inform the agent about preferred Internet sources by specifying respective RSS feeds through the Web-UI. The agent will then monitor the feeds and alert the user when encountering relevant information making use of a Jabber client.

In the broad study (S4.2) the users are left much more unguided. Other than in S4.1 where the users are confined to the two domain-specific interfaces and the Web-UI, in S4.2 they have access to all UIs. The users are more generally informed that the agents have some domian-specific knowledge about popular music and besides can access the Internet. Depending on what information the users frequently access, which Internet sources they specify, and which feedback (praise, scolding) they give to their agents, the agents will adapt to the users' interests and aim at providing more related information. The more effort the users invest in training their agents, the better the agents should become in providing the users with new information relevant to the users' interests.

6.2 Evaluation Example: Use of Cognitive Aspects in an RSS Feed Filtering Scenario

We make use of a simple scenario for RSS feed filtering to illustrate the platform's capability to augment the agents' abilities by incorporating cognitive aspects into processing. Since the different agents can exist at the same time in the same environment and thus be subject to the same external influences, they can be reliably compared and evaluated.

Recall: In the RSS Feed Filtering scenario, the agents' task is to provide their users with (potentially) interesting information gathered from music-related RSS feeds on the Internet. The users train their agents according to their interests, thus creating an agent profile containing relevant keywords such as artist names, song names and genre names. The users also provide RSS feed URLs to their agents, which the agents then query for new information.

The research question for this simple scenario is whether certain properties of an agent improve the overall user satisfaction (as measured by the rate of false positives and negatives). Therefore, in addition to the basic keyword-based filtering of RSS feeds, the following elements can be added to a particular agent:

- The ability to find similar agents (agents with a similar

agent profile) and consider those agents' RSS feeds in addition to the ones provided by its own user. This essentially extends the agent's search space.

- The ability to find items (artists, songs, etc.) similar to the ones specified in the agent profile. These items are then added to the filter keywords, thus extending the agent's search criteria.

The services required to perform these tasks (finding similar agents, finding similar artists, etc.) are provided by the RASCALLI framework and made available to the agents as additional Tools in the action and perception layers.

This setup allows us to create and compare four agent definitions, containing neither of the two features, one of them, or both. For the evaluation, different users are equipped with one of the available types of agents. The users can then identify false positives and negatives.

Obviously, this example could be implemented without a complex system such as the RASCALLI platform. However, even in this simple case the platform has some advantages to offer:

- Components can be shared between agents. For example, the service to find similar agents is an external web service, which is made available within the platform as a simple OSGi/Java service.

- The platform provides a single runtime environment for all the agents (there would be multiple agents of each kind), so that inter-agent communication (for exchanging RSS feed URLs) can be easily implemented.

7. CONCLUSIONS

The RASCALLI platform meets a unique set of requirements, that is not targeted by any of the investigated platforms or methodologies. In particular, it supports the development and execution of multiple agents of different kinds. Furthermore, it supports the evaluation and comparison of such different agents within a single environment.

Future work includes the completion of the development environment (e.g. the integration with development tools such as the Eclipse IDE), as well as the possible integration of certain aspects of other projects, such as BOD, AKIRA or JADE.

The RASCALLI platform and selected system components will be made available to the research community by the end of 2008 via the project homepage.[12] In particular, the project partners Research Studios Austria (SAT) and the Austrian Research Institute for Artificial Intelligence (OFAI) will provide an open source version of the platform and Tools, respectively. In addition, SAT will make available their other system components and user interfaces as managed services with limited support and data volume free of charge for the research community. The German Research Center for Artificial Intelligence (DFKI) will make available their data sources via an open research license and provide their system components as webservices. The companies Ontotext and Radon Labs will make available their SwiftOWLIM semantic repository and the 3D client, respectively. The New Bulgarian University (NBU) will conribute their DUAL-inspired implementation of an agent's mind.

[12]http://www.ofai.at/rascalli

8. ACKNOWLEDGMENTS

This research is supported by the EC Cognitive Systems Project FP6-IST-027596-2004 RASCALLI, by the national FFG program 'Advanced Knowledge Technologies: Grounding, Fusion, Applications' (project SELP), and by the Federal Ministry of Economics and Labour of the Republic of Austria. The work presented here builds on joint work within the RASCALLI project. In particular the authors would like to thank our collegues from the following institutions: SAT, OFAI, Radon, NBU, Ontotext and DFKI. Finally, our participation in the PerMIS'08 workshop is supported by the European Network for the Advancement of Artificial Cognitive Systems (euCognition, European Commission, Unit E5 - Cognition, FP6 Project 26408).

9. REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE: a FIPA2000 compliant agent development environment. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 216–217, New York, NY, USA, 2001. ACM.

[2] J. Bryson. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, MIT, Department of EECS, Cambridge, MA, 2001. AI Technical Report 2001-003.

[3] J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors. *Embodied Conversational Agents*. MIT Press, 2000.

[4] E. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630, 1957.

[5] B. Kokinov. A hybrid model of reasoning by analogy. In K. Holyoak and J. Barnden, editors, *Analogical Connections*, volume 2 of *Advances in Connectionist and Neural Computation Theory*, pages 247–320. Ablex, 1994.

[6] B. Kokinov and A. Petrov. Integration of memory and reasoning in analogy-making: the AMBR model. In D. Gentner, K. Holyoak, and B. Kokinov, editors, *Analogy: Perspectives from Cognitive Science*. MIT Press, in press.

[7] G. Pezzulo and G. Calvi. Designing modular architectures in the framework AKIRA. *Multiagent Grid Syst.*, 3(1):65–86, 2007.

[8] M. Skowron. *A Web Based Approach to Factoid and Commonsense Knowledge Retrieval*. PhD thesis, Hokkaido University, Sapporo, Japan, 2005.

[9] M. Skowron and K. Araki. Effectiveness of combined features for machine learning based question classification. *Special Issue of the Journal of the Natural Language Processing Society Japan on Question Answering and Automatic Summarization*, 12(6):63–83, 2005.

[10] M. Skowron, J. Irran, and B. Krenn. Computational framework for and the realization of cognitive agents providing intelligent assistance capabilities. In *Proceedings of the Cognitive Robotics Workshop at the 18th European Conference on Artificial Intelligence*, 2008.