The RASCALLI Platform – For a Flexible and Distributed Development of Virtual Systems Augmented with Cognition

Brigitte Krenn, Christian Schollum[1]

Abstract—**We introduce a modular and flexible development environment (the RASCALLI platform) for modelling and realization of artificial systems that are augmented with various aspects of cognition. The platform supports a distributed development of system components and the integration of components from different areas of research realized in a variety of programming languages. It supports multi-agent architectures and multi-agent instances, as well as multiple versions of the same component(s) to run in parallel on a single platform instance. Individual agents are created in a Lego-like manner via assembly of components deployed in the platform.**

The current application domain are smart companions in the popular music domain that assist their users in retrieving personalized, domain-specific information from the Internet and other knowledge sources. The system components and services currently available within the platform are a starting point for the development of and experimentation with various kinds of virtual systems, implementing different aspects of cognition.

I. Introduction and Motivation

THIS document introduces a platform and basic set of system components for the creation of so called RASCALLI agents (Responsive Artificial Situated Cognitive Agents that Live and Learn on the Internet), henceforth Rascalli for the plural, and Rascallo or Rascalla for the singular, depending on whether the agent is represented by means of a male or female character in the user interface. At the time of writing, only the male interface character is fully developed. Thus we will use Rascallo to refer to an individual RASCALLI agent.

With the RASCALLI platform, we want to provide a modular and flexible development environment for modelling and realization of artificial systems that are augmented with various aspects of cognition. We do this, because on the one hand we want to explore if and which aspects of cognitive modelling may improve systems that pursue complex information processing and communication tasks, and that need to adapt to the user. On the other hand, developing such systems in a flexible and time saving way will foster understanding of what an artificial cognitive system might be. The focus of the paper lies on the design and implementation of the platform.

The euCognition Web site collects answers to the question what cognition, what a cognitive system is.[2] The answers stem from euCognition members covering a broad range of European researchers in the fields of natural and artificial cognitive systems. The given answers illustrate in a catchy way the diversity of approaches to cognitive systems and the early stage of research in artificial cognitive systems. In this respect we see the RASCALLI platform, and the system components and services available within the platform as a starting point for the development of and experimentation with various kinds of Rascalli, implementing different aspects of cognitive systems. The platform supports flexible creation of individual Rascalli via assembly from a pool of existing components, and allows multiple Rascalli to run on a single platform instance. This way different incarnations of Rascalli can be exposed at the same time to the same environmental conditions, and thus develop in parallel. They then can be evaluated according to what information they have gathered, what outputs they have produced to certain tasks, and so forth, and also what the user satisfaction with the different systems is. All in all, a variety of criteria can be defined to identify a winning system. Diagnoses can be made about which aspects of cognitive modelling are successful in the given application domain, and under the given conditions. If the cognitive aspects modelled in individual successful Rascalli conceptually fit together, the platform technically supports the creation of a new incarnation that combines the features of the individual ones, and the assessment can start anew. So far for the motivation of the RASCALLI endeavour.

Cognitive aspects of relevance for the Rascalli are: Learning: Rascalli learn through interaction with a (complex) environment consisting of the Internet, of domain-specific knowledge bases, of other Rascalli, and last but not least of users. Each Rascallo has a single user, one user, however, may have several Rascalli. Adaptation:

Manuscript received March 14, 2008. This work is supported in part by the EC Cognitive Systems Project IST-027596-2004 RASCALLI (Jan. 2006 – Dec. 2008). The Austrian Research Institute for Artificial Intelligence and the Research Studio are supported by the Federal Ministry of Economics and Labour of the Republic of Austria.

Brigitte Krenn is with the Austrian Research Institute for Artificial Intelligence, 1010 Vienna, Austria (phone: +43 1 5324621-2; fax: +43 1 5324621-9; e-mail: brigitte.krenn@ofai.at). Since spring 2006 she is also with the Research Studio Smart Agent Technologies, 1160 Vienna, Austria.

Christian Schollum is with the Research Studio Smart Agent Technologies, 1160 Vienna, Austria (e-mail: christian.schollum@researchstudio.at). The work on the RASCALLI platform architecture is part of Christian Schollum's diploma thesis.

[1] The authors appear in alphabetical order.

[2] http://www.eucognition.org/wiki/index.php?title=Definitions_of_Cognition

Rascalli use observed evidence to change their behaviour. In particular they log user interactions with the system such as positive and negative feedback, how the users explore domain-specific tools, and which information the users explicitly provide to the system, e.g. via URLs and RRS feeds. These are the raw data on which different adaptation mechanisms may operate. Perception and action: Rascalli relate perception and action in a meaningful way by employing models of experience and learning, reasoning and memory, etc.

The paper is organized as follows: We start with a presentation of the application domain chosen (Section II). Then we discuss the platform architecture and its technical realization (Section III). This is followed by a high-level view on the architecture of an individual Rascallo and a description of current realizations of Rascalli (Section IV), and an example of how to implement a specific RASCALLI agent (Section V). The paper is rounded up by a discussion of related work and reflections on future directions for the RASCALLI platform (Section VI).

II. Application Domain: Rascalli as Smart Companions / Music Rascalli

As regards the application domain, Rascalli are smart companions, i.e., systems that support the user with information by answering user questions and by learning about user preferences and interests, and using this knowledge to present the user with new information that is related to those preferences and interests. The outside world (environment) a Rascallo has to deal with consists of: external knowledge sources, in particular the Internet and some domain-specific knowledge bases; the user, each Rascallo has a single user, whereas one user may have several Rascalli; and other Rascalli.

The current application domain is popular music. Apart from the Internet, the Rascalli have access to the following two domain-specific knowledge resources: (1) a database with up to 150 000 tracks represented via meta-information such as track name, album name, genre, artist or group name; (2) a database with personal information of artists or groups such a their real name, their marital status, hair and eye color, names of family members etc. While the former is an excerpt of a commercial database for music download also including 30-second sound snippets of each track; the latter is automatically retrieved and constantly updated from the Internet starting out from the artist names in the commercial DB and from domain-specific seed relations, cf. [1].

Suitable to the specific environment, the most important processing tools available to the Rascalli are: a collection of natural language analysis tools for processing the textual input from the user to the system (the Rascallo), as well as for extracting information from text documents retrieved from the Internet; tools for natural language querying of the domain-specific knowledge bases; an open-domain question answering system through which the Rascalli can retrieve arbitrary information from the Internet, of course with the risk that the retrieved information is erroneous; a multimodal natural language generation component that produces the information necessary for the Rascallo to communicate to the user via an Embodied Conversational Agent interface.

Rascalli communicate with the user in an intuitive, low barrier way by means of two kinds of user interfaces: The one interface, as already mentioned, is realized by means of an Embodied Conversational Agent (ECA,; the term has been coined in [2]). An ECA communicates to the user displaying verbal and nonverbal behaviour, i.e. synthesized speech, facial expression, gesture, posture. An example of an ECA interface of a Music Rascallo is shown in Fig. 1. Each Rascallo runs on its own 3D client at the user's computer. The other interface is a Web interface where the Rascallo can present the user with links to Web pages of interest and all other information for the user that is too long or too clumsy to be well presented via speech, such as longer enumerations, pieces of text that cover more than a few sentences, sequences of digits and characters, etc.

Rascalli show reactive behaviour when they respond to user queries, and they show proactive behaviour when they autonomously crawl the Internet to find information suitable for the user, and when they contact the user for instance via email, or using a Jabber client.

Fig. 1. Music Rascallo: ECA interface.

The user can pose domain-specific questions to the Rascallo via a text input window in the 3D client or in the Web interface. She can give positive or negative feedback to the answers given by the Rascallo via praise and scolding buttons. In addition, she can explicitly make known URLs and RSS feeds of her interest to her Rascallo. This is done via the Web interface. A further possibility for the user to profile her Music Rascallo according to her interests is via two applications that allow browsing the music and artist

data.[3] The user activities within the applications are tracked by the Rascallo and used as input to learning and adaptation.

III. RASCALLI PLATFORM: ARCHITECTURE AND REALIZATION

A. Requirements

The design of the software architecture of the RASCALLI platform is determined by a number of requirements that will be discussed in the following. They stem from the wish to run multiple Rascalli on a single platform, and the needs to provide a platform that supports a plug and play approach, as well as the realization of and experimentation with different cognitive architectures/models/components, and that allows for distributed development of individual components. We also particularly aim at a research setting, where the integration of existing components is preferable to re-implementation, and where system integration is likely to be a technically non-trivial task.

System Integration: In a research project like RASCALLI where components from a variety of research areas must be integrated, existing components are often re-used and assembled to new configurations for new purposes. Since they are implemented with vastly different technologies, the integration task is usually non-trivial and a significant amount of work is needed to set up and maintain a complete environment for each researcher. Therefore we aim at providing a single environment that can be shared by multiple researchers. The RASCALLI platform provides such an environment, where existing components need to be integrated only once to be available.

Component-based Architecture: All the parts that are required to set up a specific Rascallo are developed in a component-based fashion so that individual Rascalli can be assembled from these components in a Lego-like manner [3]. The components are separately deployed to the platform such that a particular realization/incarnation of a Rascallo that is built entirely on deployed components can be assembled by configuration only.

Distributed Development: Implementing systems that are able to combine components from such different fields of research as it is the case with the Rascalli, requires the development of components in an independent, distributed fashion. Technically this is achieved by employing a component-based approach and loose coupling between application layer components. The major conceptual task is developing a shared ontology between the components so that they can talk to each other. This ontology describes the input and output data of each component. New components have to describe their input and output data in terms of the shared ontology. It should be clear that the platform mechanisms are only technical means to support flexible assembly of Rascalli instances.

Multi-agent/Multi-architecture: The platform supports multiple Rascalli running at the same time. These may be Rascalli with the same mind-tools configuration, as well as systems of different configurations, possibly with conflicting versions of tools or minds, other central control components, or architectures supporting emergent behaviour. The platform thus allows vastly different system organisations to be implemented and run in parallel, some of which not necessarily have anything in common with human-like cognition.

Multi-user: This is an extension of the multi-agent requirement. Since each agent has its own user, the platform must allow multiple users to connect at the same time.

Multi-version: In a distributed development process, multiple versions of the same components are used at the same time. This is because different developers work on different parts of the system, while making use of stable versions of other parts.

B. Realization

In this section, we give an overview of the technical realization of the platform, which is implemented as a three-layered architecture (Fig. 2). In the following, we briefly address the different layers.

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)

Fig. 2. RASCALLI platform layers.

Infrastructure Layer: The infrastructure layer contains basic tools and components used in the RASCALLI project. Specifically, these are Java[4], Maven[5] and OSGi[6]. 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.

Framework Layer: The framework layer comprises general platform services and utilities employed by the Rascalli, including networking support (TCP, HTTP), synchronization and concurrency support, logging support, email support, RDF handling support, technology integration support (Perl, web services, etc.), and Spring support.

Agent Layer: The agent layer is the application layer of the platform and contains the components of the actual agents. This layer consists of multiple sub-layers, which are shown in Fig. 3 and will be briefly described in the following.

[3] See http://musicexplorer.researchstudio.at and http://rascalli.dfki.de/ontology/, respectively.

[4] http://java.sun.com/

[5] http://maven.apache.org/

[6] http://www.osgi.org/

Agent Architecture Layer: The RASCALLI platform supports the implementation of agents of different agent architectures. This layer is used to define the agent architectures. An agent architecture, in this context, is a blueprint which defines the architectural core of the particular type of Rascalli. For instance, all of our current implementations of individual Rascalli are built upon a Mind-Body-Environment (MBE) architecture. (See Section IV for details.) More precisely, an agent architecture defines 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.

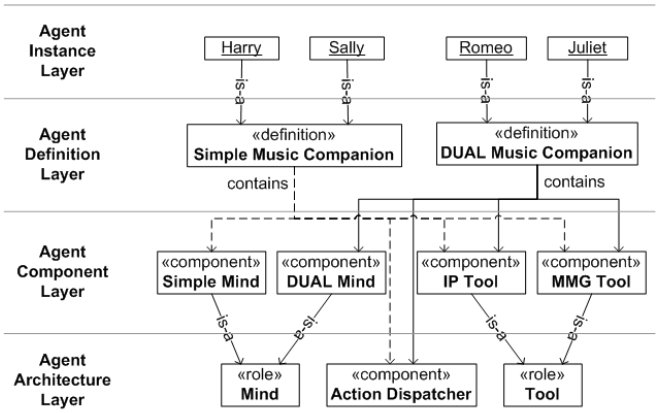

Fig. 3. The Agent Layer, consisting of four sub-layers.

Agent Component Layer: The 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 Components Layer of a specific agent architecture. Different agent definitions for the same agent architecture might contain different components for certain roles.

For example, based on the MBE-architecture three different Rascalli configurations are currently developed and implemented, namely: simple minded Rascalli, Rascalli utilizing an implementation of the DUAL cognitive architecture as their central control unit, and Rascalli realizing an approach to affordance-based learning. Thus an agent definition for the MBE-architecture might include the DUAL-Mind while another definition includes the Simple Mind and another one the Affordance-based Mind. Note that different agent definitions might also include different versions of the same component.

Agent Instance Layer: This layer contains the individual Rascalli instances. Each Rascallo is an instantiation of a specific agent definition. Note that individual Rascalli might have additional configuration beyond what is provided on the Agent Definition Layer. For example, each Rascallo might be given a name. It will also need to know about its owner/user, etc. The Rascalli instances Harry and Sally, for instance, implement the same agent definition (simple minded music companion). Romeo and Juliet, on the other hand, are equipped with a DUAL mind. All four make use of the input processing tool (IP Tool) and the multimodal generation component (MMG Tool), Fig. 3.

IV. Examples for Rascalli Building upon a Mind-Body-Environment Architecture

In Fig. 4, we present a high-level view on the Mind-Body-Environment architecture underlying an individual Rascallo. The architecture combines a central control unit (the mind), and a perception/action layer (realized by means of processing tools that may function as sensors and effectors; cf. the processing tools in Section II) which mediates between the mind and the environment. The mind may be a very simplistic component or a rather complex system with a variety of components.

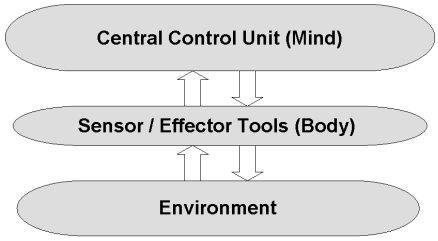

Fig. 4. RASCALLI MBE architecture. A high level view of the RASCALLI architecture distinguishing the mind, the body (sensor and effector tools), and the environment.

For proof of concept, we have realized three types of Rascalli which differ in their mind component. The one is equipped with a simple mind. The other one realizes memory structuring employing the DUAL/AMBR cognitive model, and the third one implements an affordance-based approach to learning of action selection. All three types of Rascalli operate on the same kind of environment, have access to the same domain-specific knowledge bases (currently from the music domain), and use similar tools for perception and action.

A. A Simple Mind

The Simple Mind is a trivial implementation of the mind component, based on hard-coded action selection rules. These rules look for specific cues in the input data arriving from sensor tools, extract relevant information and pass this information on to the appropriate effector tool. Even though seemingly non-trivial behaviour 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.

B. Memory Structuring using the DUAL/AMBR Cognitive Model

Another implementation of the cognitive architecture behind a Rascallo's mind is based on the cognitive model DUAL/AMBR, [5], [6]. An existing Lisp implementation has been adapted to be integrated with the RASCALLI platform. In order to do so, the DUAL/AMBR model has

been adapted to the human-like metaphor of Mind-Body-Environment and to the specific knowledge structures and tasks of the Music Rascalli application. The focus of the mind model lies on memory structuring, distinguishing between a long term memory (LTM) where general and episodic knowledge is stored, and a working memory (WM) representing the active part of the LTM. The mind has a frame-based representation of selected aspects of the environment and experiential memories related to the specific, episode-like interactions of the Rascallo with its environment.

From a technical point of view, the communication between the mind and the body tools is a translation of the RDF[7] output of the tools into frame-based representations internal to the mind and the output of the mind into RDF input to the tools.

C. An Affordance-based Approach to Learning

In this approach, insights from affordance-based research initiated by Gibson [7] and from affordance-based research in robotics [8] are being adapted to the design of the cognitive model of the RASCALLI agents and their interaction with the environment. In particular, the affordance-based approach to learning as described in [9] and [10] was transferred from the physical robotics environment, it has initially been designed for, to the virtual environment of the Rascalli.

The goal is to develop Rascalli whose behaviour is biologically inspired and who gain knowledge - clearly different from human knowledge, but grounded in the theory of human use of affordances. This kind of knowledge first of all forms the basis for action selection within a Rascallo, but in the future will also be exploited for modelling information exchange between Rascalli, as well as for communication between the Rascalli and their users.

V. How to Implement an Agent

This section briefly describes the process of implementing a new Rascallo within the RASCALLI platform. Our example is based on the implementation of the Simple Minded Rascallo.

Agent architecture: First, the agent architecture must be defined. An agent architecture is deployed to the platform as a separate component. This allows us to deploy multiple agent architectures as well as multiple versions of the same architecture concurrently. Since the platform is based on Maven and OSGi, each component is implemented as a separate Maven project and deployed to the platform as an OSGi bundle.

An agent architecture defines the roles of a Rascallo's components. For this example, we define two roles: Mind and Tool. These are defined as Java interfaces, to be implemented by concrete components on the agent component layer. In addition to the roles, the architecture also implements components common to all agent definitions. In this case, we need a component that dispatches output from the mind to the appropriate tools.

Finally, we implement a factory, which is later used by the platform to assemble individual Rascalli. This factory knows, for example, how many components of each role are required (e.g. exactly one mind, but multiple tools) and how they are to be combined into a functioning Rascallo.

Agent components: After creating the architecture, we need some implementations of the defined roles, in our case the components:

- Simple Mind
- Input Processing Tool
- Multimodal Natural Language Generation Tool
- Question Answering Tool

Each of these components is implemented as a separate Maven project and deployed to the platform as an OSGi bundle.

Agent definition: Now that the necessary components are implemented, the specific assembly of the components must be defined which later can be instantiated by the platform. Our Simple Minded Rascallo consists of all the components just implemented. However, if we later choose to implement an alternative mind implementation, we can create another agent definition which contains the new mind component instead of the simple one.

An agent definition is also implemented as a separate maven project and deployed as an OSGi bundle. Deploying the agent definition as a separate bundle is necessary, since different agent definitions may contain the same agent component(s) in different versions.

Agent instance: The platform provides a user interface for creating new instances of available agent definitions. Each agent instance requires additional configuration, such as the agent's name and user, which must be provided upon agent creation.

VI. Related Work and Future Directions

The major standards organization in the area of Agent Oriented Software Engineering (AOSE) is FIPA[8], which is concerned with the standardization and interoperability of multi-agent systems. One example of these efforts is FIPA's Agent Communication Language (ACL).[9]

Of the large number of FIPA compliant agent platforms, JADE (Java Agent Development Framework)[10] is one of the more important ones. JADE is a Java-based middleware for multi-agent systems, with a strong focus on agent communication. It supports the implementation of distributed systems running on hardware platforms ranging from enterprise servers to hand-held devices.

[7] http://www.w3.org/RDF/

[8] http://www.fipa.org/

[9] http://www.fipa.org/repository/aclspecs.html

[10] http://jade.tilab.com/

While JADE and similar FIPA compliant agent platforms offer a strong middleware layer for distributed multi-agent systems, including agent life-cycle management, mobile agents and agent communication, as well as rich graphical tools for agent development, they do not meet major requirements of the RASCALLI platform. While the RASCALLI platform supports the execution of multiple agents, it is not a multi-agent system in the traditional sense, where agents are independent components of a larger application (possibly spread across a distributed system). Instead, Rascalli are complete individual entities that simply happen to 'live' in the same environment and are intended to communicate with each other. Furthermore, none of the aforementioned 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. This development style is specifically geared towards research projects experimenting with alternative cognitive architectures, and combining them with a variety of processing and generation tools as it is the case with the tool set the Music Rascalli are equipped with.

The RASCALLI approach also differs from existing software systems for cognitive modelling such as AKIRA [4][11] or AmonI[12]. While the latter two provide specific means for modelling cognitive processes, the RASCALLI platform is a more general framework for implementing a variety of different models and architectures.

Thus a direction of further development for the RASCALLI platform which suggests itself is the integration of such dedicated components for modelling cognitive aspects, as well as the implementation of selected FIPA standards (e.g. ACL) or the integration with one of the existing FIPA compliant agent platforms. Another promising future development of the platform is to interface it with robot platforms, and thus bring together artificial systems designed to be part of the physical world with systems capable of communicating and adapting to the user, and capable of complex information processing in dynamic virtual environments such as the Internet.

ACKNOWLEDGMENTS

The authors thank Gregor Sieber and Marcin Skowron for providing the sensor and effector tools; Stefan Kostadinov and Maurice Grinberg for integrating the DUAL/AMBR model, Jörg Irran for transferring the affordance-based learning model developed in the EU Cognitive Systems project MACS to RASCALLI and Marcin Skowron for an initial implementation of action selection, Peter Hlavac and David Mann for the music browser, and Xiwen Cheng and Feiyu Xu for the artist data browser, and last but not least the team of Radon Labs for implementing the 3D client and the Music Rascallo interface.

The work on the RASCALLI platform architecture and the platform implementation are part of Christian Schollum's diploma thesis.

REFERENCES

[1] F. Xu, H. Uszkoreit, and H. Li. A Seed-driven Bottom-up Machine Learning Framework for Extracting Relations of Various Complexity. In Proceedings of ACL 2007, Prague, 2007.

[2] J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, (eds). Embodied Conversational Agents. MIT Press, 2000.

[3] G. T. Heineman and W. T. Councill. Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Professional, Reading, 2001.

[4] Giovanni Pezzulo and Gianguglielmo Calvi (2007). Designing Modular Architectures in the Framework AKIRA. Multiagent and Grid Systems, 3, 65--86.

[5] B. Kokinov. A hybrid model of reasoning by analogy. In K. Holyoak and J. Barnden, eds. Advances in connectionist and neural computation theory: Vol.2. Analogical connections (Chapter 5, pp. 247- 318). Norwood, NJ: Ablex, 1994.

[6] M. Grinberg and B. Kokinov. Simulation of Episode Blending in the AMBR Model. In: Proceedings of the 4th European Cognitive Science Conference. Erlbaum, Hillsdale, NJ, 2003.

[7] J. J. Gibson. The ecological approach to visual perception. Hillsdale, New Jersey; London: Laurence Erlbaum Associates, 1986.

[8] F. Kintzler et al. Affordance-related Robotics Research – A Survey. Journal for research on adaptive behaviour in animals and autonomous, artificial systems (submitted 2007).

[9] G. Dorffner, J. Irran, F. Kintzler, P. Pölz. Robotic learning architecture that can be taught by manually putting the robot through action sequences. FP6-004381 Project MACS Deliverable 5.3.1. 2005.

[10] J. Irran, F. Kintzler, and P. Pölz. Grounding Affordances. In proceedings: Trappl R. (ed.): Cybernetics and Systems 2006, Vienna: Austrian Society for Cybernetic Studies, 2006.

[11] https://sourceforge.net/projects/a-k-i-r-a/

[12] http://www.cs.bath.ac.uk/ai/AmonI-sw.html