System Integration Supporting Evolutionary Development and Design

Thorsten P. Spexard and Marc Hanheide

Abstract. With robotic systems entering our daily life, they have to become more flexible and subsuming a multitude of abilities in one single integrated system. Subsequently an increased extensibility of the robots' system architectures is needed. The goal is to facilitate a long-time evolution of the integrated system in-line with the scientific progress on the algorithmic level. In this paper we present an approach developed for an event-driven robot architecture, focussing on the coordination and interplay of new abilities and components. Appropriate timing, sequencing strategies, execution guaranties, and process flow synchronization are taken into account to allow appropriate arbitration and interaction between components as well as between the integrated system and the user. The presented approach features dynamic reconfiguration and global coordination based on simple production rules. These are applied first time in conjunction with flexible representations in global memory spaces and an event-driven architecture. As a result a highly adaptive robot control compared to alternative approaches is achieved, allowing system modification during runtime even within complex interactive human-robot scenarios.

1 Introduction

Robots are developing from very specific tools towards generic and flexible assistants or even companions. With the applicable fields getting broader and the abilities of the robots increasing, it is simply not feasible to construct a robot system from scratch. In contrary, a robotic system should evolve and successively incorporate new abilities and functions as soon as they are available and needed. However, robotic research is still far away from its maturity and still highly dynamic. Thus, an integration approach that paces up with the fast development in robotics, needs to

Thorsten P. Spexard · Marc Hanheide
Applied Informatics, Bielefeld University, Germany
e-mail: {tspexard,mhanheid}@techfak.uni-bielefeld.de

embrace change and be flexible in its particular integration strategy. We present our approach focussing on the techniques to create robotic systems which fulfill these requirements resulting in an *Evolutionary System Integration Architecture ESIA*, which evolves with new demands. Our general focus thereby lies on *interactive* robots, that feature sets of different behaviors and abilities. Though it is self-evident that for any successful software system a detailed architecture has to be designed in advance, reality shows that during software integration often unforeseen challenges occur and adaptations have to be made quickly. Thus ESIA supports simple and fast, well-structured mechanisms for system modifications without creating an inscrutable kludge. Since in the scientific community larger-scale robotic systems are constructed in joint efforts, like in the European Integrated Projects CoSy [5] and Cogniron [12]), system integration can also pose a social challenge. Thus, aspects such as "informational value" and "familiarization time" are relevant as well. This integration of previous independently developed robot functions into one system in rather short time frames can be seen as a typical problem of legacy integration.

To accept this challenge our evolving integration approach makes use of a software architecture proposed by Wrede as information-driven integration (IDI) [13]. It combines benefits of event-driven architecture and space-based collaboration applying flexible representation of information using the XML data set. Though the IDI approach indeed serves as a foundation to evolutionary system integration with respect to interface changes and flexible orchestration, it does not strive to provide a comprehensive solution to the question "How to coordinate components and how to design their interplay?". In our presented approach we strive for a generic solution to this challenge, which shall (i) maintain a highly decoupled system architecture while (ii) still allowing flexible control and arbitration necessary for the ever changing requirements of different robotic functions. A central organizer is used to coordinate the separate module operations to the desired system behavior regarding both, serialization, and arbitration of the different module actions. In contrast to usual system controllers or planners the organizer is not directly connected with the system components but simply modifies the memory content based on rules in such a way that the desired behavior emerges.

2 Related Work

Derived from classical AI a rule-production-system (RPS) uses a set of rules, a working memory, and an interpreter to solve tasks. Rules consist of requirements which have to be fulfilled so that certain actions take place. For the evaluation whether a condition is satisfied the data within the working memory is used. The robot Caesar is using RPS [11] by a rule based controller, set on the top of a modular system with multiple layers of different abstraction levels. In contrast to ESIA only on the bottom level standard programming language is used for sensor and actuator drivers. For all remaining modules Erlang is used as demonstrator specific runtime environment. The module information is read once on start up and is based on fixed interface descriptions. Thus algorithms created in different and more public programming languages have to be re-implemented especially on the basis of the system specific interface needed by the controller. We propose the support of various widespread programming languages like C(++) or Java in combination with a generic underlying data model, e.g., XML representation.

Another aspect is the flexibility of rule-production-systems during runtime. Though easily adaptable when the system is shut down Ishida discusses in [7] the opportunity in adapting the behavior during runtime. A meta programming language is used to dynamically change the rules themselves while the system works. We adapted these ideas by assigning the rule evaluation to separate threads with one thread per rule and developing a way to dynamically change rules during runtime. Instead using another programming language, or more general another representation, we propose the idea of joining the representation of behavior rules and the data they apply to. Furthermore the rules are no longer separated from the working memory and set on the top of the system, but are memory content which is equally treated as, e.g., processed sensor data.

The rule definition complies with the Event-Condition-Action model introduced by [4]. To satisfy the requirement of a rule, so that the associated action takes place, not only a certain event has to take place, but additionally the current system situation is considered by an additional condition which has to be complied with. Using the system state in combination with a recent event, involves the handling of temporal dependencies since the events leading to the desired system state had to take place prior to the event finally triggering the action execution of a rule. Although the importance of temporal dependencies is described in [10], temporal aspects in evaluating rules are only referred to in terms of a given global execution time which has to be achieved. The question, how to deal with, e.g., temporal concurrency in parallel systems is left open.

3 Creating Systems for HRI

We implemented the presented approach on two different robotic demonstrators focussing especially on extended abilities for advanced human-robot interaction. One demonstrator is BIRON as depicted in Fig. 1(a), which is used in a Home-Tour scenario [9]. In this scenario naive users have to guide the robot around in a real world flat to familiarize the robot with a new vicinity. The users receive only a brief introduction to the system and have to intuitively interact with it by naturally spoken dialog and gestures. The results taken from the user interactions have to be simply integrated into the existing robotic system for continuous improvement. Additionally to system modifications on a single type of demonstrator in one scenario ESIA adapts to alternate demonstrators and scenarios. In a second scenario we use the anthropomorphic robot BARTHOC (see Fig. 1(b)) in a receptionists setting [2]. In this setting robot and human oppose each other at a table with a map of the surrounding lain between them. The interaction partner refers to the map in a natural way and the

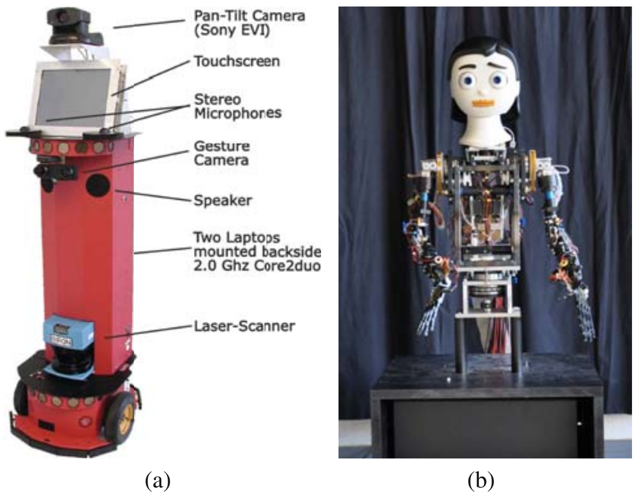

Fig. 1 The robot demonstrators BIRON (a) and BARTHOC (b). The presented approach has successfully been applied to both of them for advanced HRI.

robot reacts with the same modalities by verbal and gestural output. Taking advantage of a more human-like appearance as BIRON emotional information based on prosody taken from the user's utterances are processed and replied to by adequate facial expressions of BARTHOC [6].

Event Driven Information Exchange

To realize enhanced module communication with ESIA, the concept of information-driven integration is exploited. A module within the system represents either an event source, an event sink or even both of them. Event sources publish their information without a specific receiver, while event sinks subscribe to specific events containing information they need to filter and process [8]. Keeping the focus on the continuous development of a complex system the event-driven information exchange strongly supports the loose coupling of the system components. An event source is not directly connected of its corresponding event sinks and thus it does not matter if a sink is deactivated during runtime. Vice versa if an event source is missing it appears to the sink as if there is currently no new data.

However, though the event-based architectures provide clear advantages for modular systems, they do not directly support persistence. But persistence plays an important role especially for flexible module configuration during runtime. Consider, e.g., an event which is important for the arbitration of an hardware actuator. Since the event source is not aware of its sinks it will not notice if one is currently down, and the event sink will not know about the current arbitration result after a restart. Therefor we combine the event-driven approach with a memory [13]. Besides the

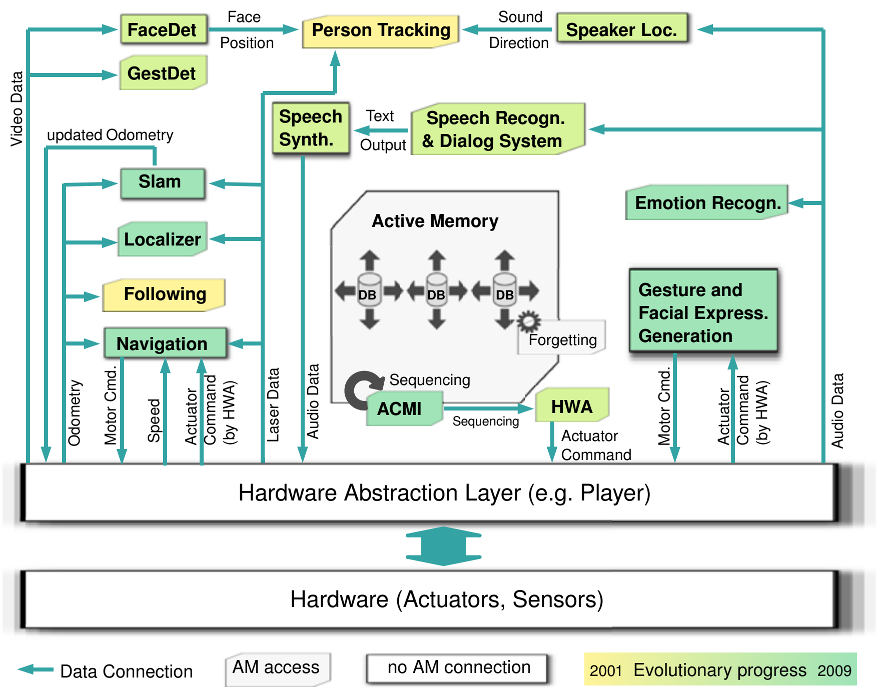

Fig. 2 ESIA for BIRON and BARTHOC: The approach enables a coordinated integration for software which has been developed over the last eight years

event notification, the event information is temporarily stored so it can simply be looked up by a restarted module to get in sync with the remaining system.

An Architecture for HRI

A more concrete description of the implemented architecture with the realized software components for the scenarios and demonstrators is shown in Fig. 2. It is easy to see that both, the reuse of domain unspecific abilities, and the exchange of domain dependent abilities are supported. While raw sensor and actuator information are passed by via direct 1:n connections, processed data is event driven exchanged by the memory, which may contain several XML databases. Using a wide spread information representation such as XML, memory entries are easy addressable by existing tools like XML Path Language (XPath). These tools are also used, to specify the events a module registers to. In principle a registration implements a database trigger, which consists of a database action. If an event takes place associated with an XML document matching the XPath and the according action, the event and the XML document are reported to the registered module. But a robot for enhanced HRI needs a synchronized global system behavior consisting of separate module behaviors to perform complex tasks like learning by interaction. Instead of adding more system knowledge to the individual modules which would have coupled them stronger together, we developed a system keeper module which interfaces with the memory, supervising the information exchange taking place.

4 Active Control Memory Interface - ACMI

One often followed control approach is to establish a direct configuration connection to any component running. This is useful when important information has to be processed under guarantee. The ACMI module supports direct control connection by implementing a generic remote method invocation interface, which has to be supported by the configured module. The interface is used to configure the arbitration of hardware components and provides immediate feedback to ensure the correct configuration before actuator commands are executed. But for a general approach this method contradicts the idea of loose coupling concerning the exchange or addiction of modules during runtime. The general operation method of ACMI is to change the memory content in a way that modules observing the memory react properly to generate the desired system behavior without direct connection to ACMI (see Fig. 3).

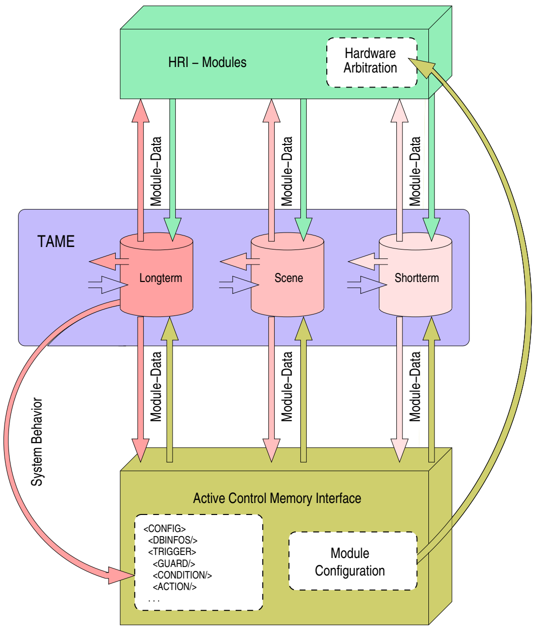

Fig. 3 ACMI operates by manipulating memory content. Exceptionally direct control via RMI is possible. Using the rule editor TAME, the operation of ACMI can be changed immediately during runtime.

Rule Interpretation and Adaptation

ACMI operates by evaluating the current memory content and thus the current system state by a set of rules. An example is given in Fig. 4. It can be compared to rule interpretors of production systems similar to ACT-R [1], but without general control connections to other system components. For the rule interpretation and appliance it reuses the same event based framework as the remaining modules. To enable a fast and easy familiarization of the developers, the amount of different rule types and their actions was reduced to a minimal set. To create full operational readiness five different actions callable by a rule were identified:

copy	copies information from one to another entry
replace	replaces information by information from another entry
remove	removes information in the memory
assert	modifies the current process status of an entry
configure	uses a memory entry as parameter for a RMI of a module

As it can easily be seen from the example a rule is represented the same way, as the information the rules apply to. This does not only keep the number of information representations a developer has to understand small, but additionally allows a kind of reuse of the framework: Instead of separating the rules which implement the system behavior from the factual knowledge, they are stored in the memory as an XML document as an important part of of the system information. Accordingly factual data derived from sensor data and procedural knowledge emerging from rule appliance are treated equally within the memory. This enables the manipulation of rules as easy as any kind of database content even during runtime. Taking into account

```
<CONFIG>
  <ACMI>
    <TRIGGER>
      <GUARD database="Scene" xpath="/SITUATION[@value=’object’]" exists="no" />
      <GUARD database="Scene" xpath="/SITUATION[@value=’localize’]" exists="no" />
      <CONDITION database="Scene" xpath="/ROOM[GENERATOR=’DLG’]/STATUS[@value=’initiated’]"
                 action="INSERT"/>
      <ACTION name="CONFIGURE">
        <SOURCE database="LongTerm" xpath="/CONFIG/HWC/LOCATING/MOVESRC"/>
      </ACTION>
      <AFFIRM value="accepted"/>
    </TRIGGER>
      ...
    <TRIGGER>
      <GUARD database="Scene" xpath="/SITUATION[@value=’object’]" exists="no" />
      <GUARD database="Scene" xpath="/SITUATION[@value=’localize’]" exists="no" />
      <CONDITION database="Scene" xpath="/ROOM/STATUS[@value=’accepted’]" action="REPLACE"/>
      <ACTION name="REPLACE">
        <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value=’localize’]"
                node="/CONFIG/SYSTEM/SITUATION[@value=’localize’]"/>
        <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
      </ACTION>
    </TRIGGER>
  </ACMI>
</CONFIG>
```

Fig. 4 Example of ACMI configuration rule and sequencing: When a room description is initially inserted by user dialog and the system is neither in a object detection nor localization task the hardware arbitration is configured to receive a new command from the localization. Subsequently the AFFIRM value is used to modify the STATUS value of the room description, which causes the next rule to set the current system state to localization and prevent other actions to take hardware control before the current task is completed.

that ACMI modifies database content, the described approach enables modification of the system not only during runtime from a human user, but also by the system itself. Thus ESIA provides a powerful basis for learning systems not only on the level of factual knowledge like objects, faces or surroundings, but also on how to behave by modifying a given basic set of rules to a more improved one according to the current operation conditions.

5 Case - Study and Conclusion

To demonstrate the advantage of the presented approach we will briefly summarize the experiences made during two integration processes conducted with partners from alternative universities located at different countries [9, 3]. Two different localization approaches based on omni-directional vision on the one hand and classical laser-based SLAM on the other hand were integrated. As preparatory work the original foreign source was remotely installed and compiled in our lab. Subsequently this integration procedure, generally applicable to ESIA was followed: First, it was determined whether the new module represented an information sink, source, or both and which process loop was used. Second, interfaces for data exchange (according general data models, modules preferably already use a non binary communication) were clarified. Third, rules depending on the outcomes of the first to steps were composed to use ACMI as adapter. Finally, it was checked for conflicts due to e.g. limited resources like actuators and arbitration as well as behavior rules were added to ACMI.

Although during the integration process it turned out that some agreements could not be realized as planned due to technical limitations or simply misunderstandings, the flexibility of the described approach allowed us to continue by simply adjusting or adding rules. As an expected result the demonstrator with its recently extended software modules was successfully tested during user studies concerning the usability of the system. It was robustly running for hours while operated by naive users. When an irritating behavior for the user occurred (in fact the robot prompted some information during interaction which should originally assist the user but in fact confuses several subjects) it was simply modified by changing behavior rules without restarting a single module. This adaptation could actually have been done even during an interaction, although it was not, to preserve experimental conditions for the test person.

The ESIA approach supported this results by providing a short integration cycle powered by a memory allowing detail introspection in data exchange, observing the system behavior by a memory editor and fixing it by changing rules online, and quick restart of single failing modules with the remaining modules continuing unaffected. This enabled us to achieve the described integration together with our partners visiting the lab in three days, both times. The positive experiences and feedback from our partners encourage us to continue in the development of ESIA as evolution is a continuous and never ending process.

Acknowledgements. This work has partially been supported by the German Research Foundation in the Collaborative Research Center (SFB 673) "Alignment in Communication" and the "German Service Robotic Initiative (DESIRE)". Furthermore the authors want to thank all the contributors for providing their time and results. Without the pleasant cooperation the creation of integrated systems would not be possible.

References

1. Anderson, J.R.: Rules of the Mind. Lawrence Erlbaum Associates Inc., Philadelphia (1993)
2. Beuter, B., Spexard, T., Lütkebohle, I., Peltason, J., Kummert, F.: Where is this? - gesture based multimodal interaction with an anthropomorphic robot. In: Proc. of Int. Conf. on Humanoid Robots, Daejeon, Korea (2008)
3. Booij, O., Kröse, B., Peltason, J., Spexard, T.P., Hanheide, M.: Moving from augmented to interactive mapping. In: Proc. of Robotics: Science and Systems Conf., Zurich (2008)
4. Goldin, D., Srinivasa, S., Srikanti, V.: Active databases as information systems. In: Proc. of Int. Database Engineering and Applications Symposium, Washington, DC, pp. 123–130 (2004)
5. Hawes, N., Wyatt, J., Sloman, A.: Exploring design space for an integrated intelligent system. In: Knowledge-Based Systems (2009)
6. Hegel, F., Spexard, T.P., Vogt, T., Horstmann, G., Wrede, B.: Playing a different imitation game: Interaction with an empathic android robot. In: Proc. of Int. Conf. on Humanoid Robots, pp. 56–61 (2006)
7. Ishida, T., Sasaki, T., Nakata, K., Fukuhara, Y.: A meta-level control architecture for production systems. Trans. on Knowl. and Data Eng. 7(1), 44–52 (1995)
8. Lütkebohle, I., Schäfer, J., Wrede, S.: Facilitating re-use by design: A filtering, transformation, and selection architecture for robotic software systems. In: Proc. of Workshop an Software Development in Robotics (2009)
9. Peltason, J., Siepmann, F.H., Spexard, T.P., Wrede, B., Hanheide, M., Topp, E.A.: Mixed-initiative in human augmented mapping. In: Proc. of Int. Conf. on Robotics and Automation (to appear)
10. Qiao, Y., Zhong, K., Wang, H., Li, X.: Developing event-condition-action rules in real-time active database. In: Proc. of Symposium on Applied computing, New York, USA, pp. 511–516 (2007)
11. Santoro, C.: An erlang framework for autonomous mobile robots. In: Proc. of SIGPLAN workshop on ERLANG, New York, USA, pp. 85–92 (2007)
12. Schmidt-Rohr, S.R., Knoop, S., Lösch, M., Dillmann, R.: A probabilistic control architecture for robust autonomy of an anthropomorphic service robot. In: International Conference on Cognitive Systems, Karlsruhe, Germany (2008)
13. Wrede, S.: An information-driven architecture for cognitive systems research. Ph.D. dissertation, Technical Faculty, Bielefeld University (2009)