An Integration Framework for Developing Interactive Robots

Jannik Fritsch and Sebastian Wrede

Applied Computer Science, Faculty of Technology, Bielefeld University, Germany
{jannik,swrede}@techfak.uni-bielefeld.de

1 Challenges in Interactive Robotics Research

In recent years there is an increasing interest in building personal robots that are capable of a human-like interaction. In addition to multi-modal interaction skills, such a robot must also be able to adapt itself to unknown environments and, therefore, it has to be capable of knowledge acquisition in a lifelong learning process. Moreover, as humans are around, reactive control of the robot's hardware is important, too. Consequently, researchers aiming to realize a personal robot have to integrate a variety of features. Due to the very different nature of the necessary capabilities, interactive robotics research is thus a truly interdisciplinary challenge.

A natural design decision is to take a modular approach to build such a complex system [Ros95]. This allows the different researchers to focus on their respective task and makes integration easier. Because integration is an iterative process in large-scale research systems, it must be possible to incorporate new modules and functionalities in an interactive robot as they become available. Such a continuous extension of the robot's functionality not only results in new or extended data structures provided by the added components but usually also requires modifications of the control flow in the integrated system. This poses questions of how data and control flow can be expressed in a way that the effort and the complexity for continuous integration remains controllable during the integration process [CS00].

As described in Chapter *Trends in Robotic Software Frameworks*, these requirements make the framework-based approach well suited to the development of software applications for Interactive Robots. In this context, a fundamental task for integration frameworks is the ability to distribute modules across different computing nodes in order to achieve the reactivity needed for human-machine-interaction [KAU04]. This applies especially to large-scale systems like personal robots. However, most researchers are no middleware experts, prohibiting the native use of, e.g., CORBA-based solutions. Consequently, the communication framework needs to provide a restricted but sufficient set of functionality that enables interdisciplinary researchers to easily integrate their components into a distributed robot system.

Knowledge acquisition and the ability to adapt to unknown situations imposes two additional functional requirements that need to be addressed. An application may need some kind of memory so data management services must be provided by the integration framework. Since adaptation and processes like e.g. attention control induce dynamics into the system, dynamic (re-)configuration of components running in the integrated system is essential, too.

In order to support the envisioned incremental development of a personal robot, not only the functional requirements *modularity*, *communication*, *module coordination*, as well as *knowledge representation and acquisition* and *dynamic (re-)configuration* must be supported by the system infrastructure. Additionally, several non-functional requirements play an important role that are discussed in the following.

Taking into account that the people carrying out research on interactive robots are usually concentrating on single topics and not the overall integration, the developed framework must be very easy to use in order to gain a wide acceptance. The approach we will outline in this chapter tries to tackle this by focusing on *simplicity* and exploiting *standards compliance*.

Another important non-functional requirement we try to address is to provide a framework that enables *rapid prototyping*. Consequently, iterative development should not only be supported for single modules but also for the integrated system. Erroneous directions in system evolution can more easily be identified if integration is performed on a regular basis starting at an early stage even when components are still missing and need to be simulated. For large-scale systems, software engineering research has shown that decoupling of modules is very important. Thus, the framework should support *low coupling* of modules. This facilitates not only independent operation of components but also minimal impact of local changes on the whole system. With a framework that adheres to low coupling, *debugging and evaluation* of a running system architecture can be supported more easily.

The contribution starts with an introduction into the concepts of the XCF framework we developed along these requirements to enable high-level integration and coordination of interactive robots. The process of robot development with this approach and the lessons learned thereby are described in Sect.s 3 and 4, respectively. A conclusion on the presented approach is drawn in Sect. 5.

2 The XCF SDK

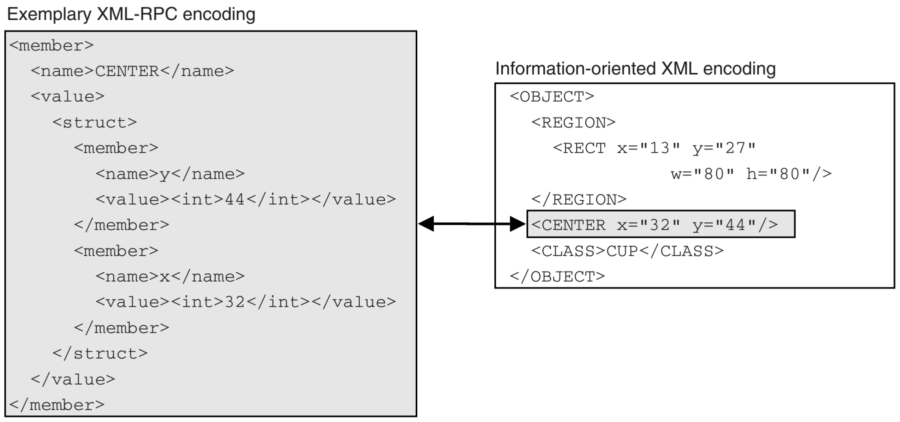
Taking into account the requirements outlined in the previous section, we developed a software development kit termed XCF consisting of a set of object-oriented class libraries and the required framework tools to develop, debug and run a distributed robotic system. The concrete implementation of the XCF concepts has been carried out in the context of interactive robotics [Cog05] and cognitive vision [Vam05] research. The complete XCF SDK consists of

- a library named XMLTIO that supports users with an API based on XPath [CD99] for simple XML processing and native data type encoding,
- the XML enabled Communication Framework (XCF) [WFBS04] itself that allows to distribute components over several computing nodes,
- the Active Memory XML server [WHBS04] for event-based coordination and data management, as well as
- the Active Memory Petri-Net Engine which allows for a declarative specification of control flow and easy development of coordination components.

The complete software is GPL-licensed and available for download at Sourceforge [Wr05]. Currently, it is available for Linux only but as all libraries used are available for other operating systems as well, porting to other platforms is possible at moderate costs. Native language bindings are written in C++, even so additional bindings are available for Java and basic XCF functionality is provided in Matlab facilitating rapid prototyping. In the following we will highlight important concepts and the resulting features of the framework that allow for efficient integration of interactive robots developed interdisciplinary.

2.1 Encoding information in XML

Since it is well-known and easy to learn as well as flexible and suited for abstract concept descriptions, the XML language was chosen as a basis to describe content transmitted, stored, and processed by the various robot modules.



```
<member>
  <name>CENTER</name>
  <value>
    <struct>
      <member>
        <name>y</name>
        <value><int>44</int></value>
      </member>
      <member>
        <name>x</name>
        <value><int>32</int></value>
      </member>
    </struct>
  </value>
</member>
```

```
<OBJECT>
  <REGION>
    <RECT x="13" y="27"
              w="80" h="80"/>
  </REGION>
  <CENTER x="32" y="44"/>
  <CLASS>CUP</CLASS>
</OBJECT>
```

Fig. 1. Contrasting XML-RPC with document/literal information encoding.

When XML is used as data exchange protocol as it is done in XML-RPC-based solutions [KAU04] it usually results in a lot of overhead through text-based representation of binary data and parameter encoding rules. Looking at the object recognition example shown in Fig. 1 you see two alternative encodings of a "`CENTER`" element. The example shows an information-oriented encoding of the center item (embedded in an object recognition result) on the right as well as a serialization of the same item in XML-RPC encoding on the left. It is not only this obvious overhead induced by a naive serialization of data to XML that is not desirable, but even worse is the loss of comprehensibility at all processing levels that is imposed by this type of encoding.[1]

Following the concept to encode information and not just data, we developed XML vocabularies for information typically exchanged in interactive robots (e.g., objects, states, events etc.). The contained semantical information is then directly selected and accessed utilizing the XMLTIO library. Declarative, name-based selection of XML nodes with XPath expressions helps in building data types that are extensible and in building systems that will not break as soon as modifications of the exchanged data structures occur. Even though this sounds trivial, carefully designed XML information structures are an important key to facilitate the overall integration progress of large scale systems developed by multiple developers.

An example that explains the benefit of the name-based selection of XML information items for system integration is depicted in Fig. 2. Four modules processing partially equivalent XML data structures are shown. This allows a component (e.g. "Hardware Control" for adjusting the robot base and the pan-tilt camera) to process information from different other system modules. Only the part in the XML document that contains information about center points ("`<CENTER>`") has to be present in an exchanged data item. Starting with a simple partial path specialization to access the context node, e.g. in this example as simple as "`/*/CENTER`", the extraction of contained information is easily possible although the context node itself might appear in varying places of different XML structures. Thus, this path expression works on both documents shown in Fig. 2. As long as no necessary information is removed, this strategy yields loose coupling and facilitates interoperability between separately developed modules.

The functionality of XMLTIO is made up of a combination of XML parsing functionality (Xerces-C) and XPath processing functionality (Pathan). Additionally, XMLTIO helps in encoding and decoding native C++ data structures into the designated XML documents and translating them back to C++ objects by exploiting template-based type lookup.

[1] The idea to directly encode information and not data in XML has recently gained more interest even in the Web Services community where SOAP-RPC encoding (which is somewhat similar to XML-RPC encoding) is being more and more replaced by the document/literal encoding we are using.

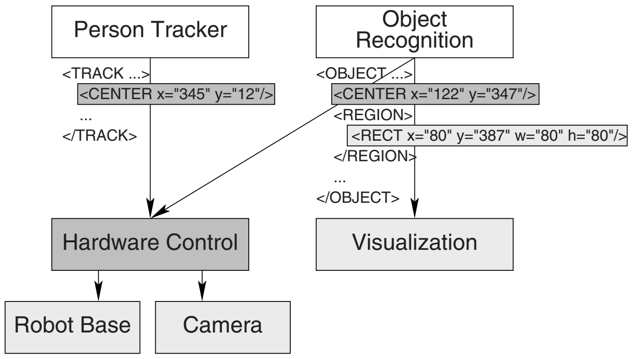


Fig. 2. Accessing common information at arbitrary locations with XPath.

2.2 Specification and Validation of Information

Meta-information, e.g., about data types, is kept separate in corresponding XML schema files and is not encoded in the instance documents themselves. Specifying data types with XML schema has several advantages in contrast to traditional programming language constructs.

First of all, the data types are independent from specific programming languages. Even so, tools for using them are available on almost every platform. Furthermore, XML schemas are able to specify content models and ranges of allowed values in great detail. Providing fine grained sets of semantically grouped declarations in separate schemas with associated XML namespaces makes them reusable throughout different systems. Complex schemas for individual modules can then easily be composed out of these basic type libraries, only adding specific complex types. If taken into account, extensibility of data types is possible with schema evolution. Even complex grammars for components capable of interpreting and validating XML documents originating from different robot modules are easy to compose and well understandable with a sophisticated schema hierarchy.

Information-oriented encoding of XML messages and the use of XML schemas for validation of the exchanged information are both very useful for system integration in interdisciplinary research projects. The focus on simple XML messages to describe exchanged information helps during project inception as almost every developer will be able to contribute to the discussion about the data flow in the system. Later on, XML grammars like XML schema allow for a rigid specification and validation of the datatypes a project consortium has agreed upon. For example, schemas have been defined in the European project COGNIRON [Cog05] to ease the integration of the partner's contributions in the realized robot prototypes. To ease this development task, XCF provides a simple command line tool that encapsulates the schema checking algorithms used in the framework for testing XML messages against the developed schemas without having to start the overall system.

2.3 XML-Enabled Module Communication

For the task of exchanging data in distributed systems a large number of different communication frameworks or middleware solutions have been proposed, ranging from message passing to RPC-style frameworks to implementations of the CORBA standard like ACE/TAO, or novel middleware frameworks like ICE. Note that, while all of the CORBA implementations aim at fulfilling the CORBA standard, most of them differ in many small but technically important aspects. Recently, much interest was awarded to XML-based communication where XML is used foremost for serialization purposes like in XML-RPC, XMPP, or SOAP (see also Sect. 2.1). To build up our integration framework we chose the ICE communications engine [Zer05] as it has a much smaller footprint than many comparable frameworks, allows high-speed data-transfer while being available as open source for different operating systems.

On top of ICE we developed a lightweight communication framework that provides only a limited but useful set of functionality that is necessary for building distributed system architectures. This XCF core library is able to efficiently exchange XML and referenced binary data (e.g., images) structures. The referenced binary data is transmitted natively in a composite object together with the corresponding XML message similar to the recently proposed XML-binary Optimized Packaging (XOP) recommendation [Wor05]. This combines the flexibility of information encoding in plain old XML documents (as explained in Sect. 2.1) with the efficiency of low-level communication semantics for large amounts of binary data.

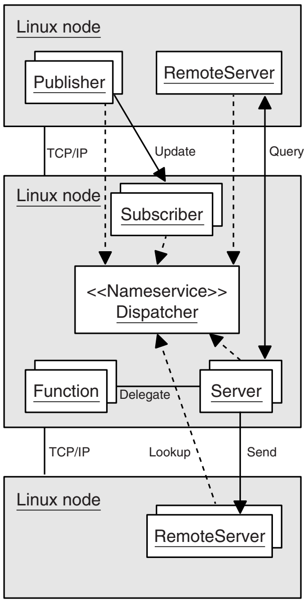


Fig. 3. Core functionality of XCF in an exemplary distributed system architecture

This XCF core features a pattern-based design and offers communication semantics like publisher/subscriber and (a)synchronous remote procedure calls/method invocations (see Fig. 3). All XCF objects and exposed methods can be dynamically registered at runtime, thus no meta-compiler is necessary. In the different communication patterns schema checking can be activated on sending and also on receiving XML data, allowing the system integrator to decide where to verify the correctness of exchanged data. As schema checking can result in a much larger computational load, it is usually switched off in normal operation after the system integration is finished.

Concurrent access of multiple clients on XCF server processes is transparently handled by message queues and exploitation of the dynamic worker thread pools provided by ICE. To achieve location transparency, we developed a name service daemon where processes register the XCF services they provide to other robot modules. This central framework component is shown in the center of Fig. 3. Thus, only the name of a component needs to be known in contrast to direct addressing using the host name of the machine it runs on. Note, that the interaction with the nameserver follows the client-dispatcher-server [BMRS96] pattern which means that after the initial name lookup, all data communication is carried out solely between the involved components. Additionally, we provide an abstract component class that exposes a generic interface for starting and stopping algorithmic processing, shutting down the whole XCF process as well as for reconfiguration and querying components during runtime of a system.

Last but not least, a central logging mechanism is supported through an XCFappenders for log4cxx,log4cpp and log4j which enable component developers to provide their log messages over the network to a central logger for debugging on a system integration level.

2.4 Event-Driven Integration and Coordination

The development of interactive applications and especially robots with sophisticated interaction capabilites imposes several constraints on a software framework for system integration and coordination. Two of the most important requirements are firstly that the system should interact with soft real-time performance and secondly the ability of the system to track the interaction context in terms of perceived episodes, events, and scenes. While the previous sections covered mainly the information representation and process coupling aspects that facilitate the building of distributed system architectures in order to ensure the reactivity needed for interaction, we will now focus on the Active Memory XML Server that forms the basis for coordination and shared data management in our integration approach.

The Active Memory concept and implementation [WWHB05] was integrated into the XCF SDK for use in our robotic framework. Since we focus on high flexibility, relational databases as backend to integrate information are infeasible. Instead, the native Berkeley DB XML database [DBX03] provides the core component of this part of the architecture shown in Fig. 4. DB XML is packaged as an embedded C++ library and provides support for transactions and multi-threaded environments. In contrast to classical database servers it offers only core features and is not a complete server application. Its therefore small footprint allows for an efficient implementation of the XML Server. Through the use of XPath, developers of robot modules can easily specify queries in a standardized and declarative manner. Additionally, runtime reindexing is supported for fast access to database contents. DB XML also allows for mixed XML and traditional data storage which is useful for binary large

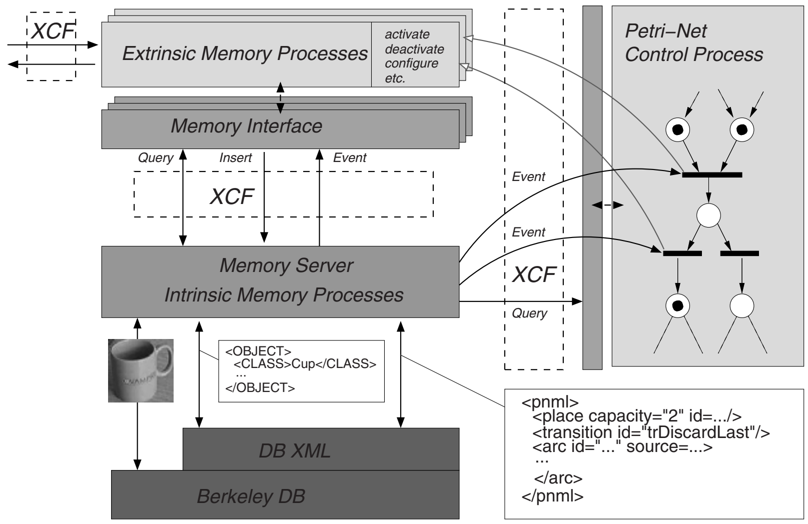


Fig. 4. Architecture for Event-Driven Integration and Coordination

objects like cropped image patches. Utilizing the reference management system we developed, binary data can be linked from and to any XML document in the database. Based on the DB XML API, the client/server architecture shown in Fig. 4 was implemented that encapsulates complexity and connects the basic insert, select and update methods to external XCF processes.

On top of this data integration approach we added an event concept which is the basis for system coordination. The general idea of the event system built on top of the native XML backend is to use XPath expressions as subscription language. For system integrators this yields a very generic and powerful registration method to couple modules of a robot architecture to specific parts of the information flow in the system. For that reason the policy of information modeling described in Sect. 2.1 is highly relevant. With the document/literal encoding of information every developer can easily express his or her interest in specific parts of system information by means of an XPath-based event subscription. To express the semantic action that is associated with a data item processed in the memory server, usually an event specification is enhanced by the specific action that is executed within the memory server on that data item, e.g. insert, query, update or delete. Whenever a basic method is called, the server instance notifies all processes that have been registered for that kind of action and the type of involved memory element. The event message includes the event source which is the matching XML document. If the notification fails, the event notification is discarded and the subscription gets removed.

Coordination is thus implicitly data-driven and *not* bound to explicit links between a fixed set of components present in the system. A simple example

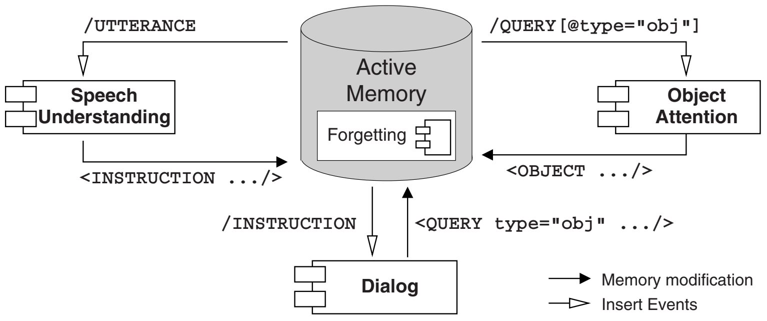


Fig. 5. An exemplary use case for event-driven integration.

of an application of this concept is shown in Fig. 5 where three components of a robot architecture are coordinated by event listeners registered on the corresponding XPaths and memory actions.

Additionally, the Active Memory XML Server features a runtime environment based on python where internal processes[2] like forgetting or information fusion can be realized that need to work on huge parts of the information mediated through an active memory instance.

To provide more complex coordination methods for multiple components operating concurrently in an interactive robot, we developed an application of petri-nets for system integration purposes. Petri-nets, in general [Pet81], extend classic state machines by the ability to represent concurrency. Thus, they are well suited for modeling structure and behavior of parallel distributed systems. The current marking of a petri-net corresponds to a specific system configuration. Dynamic changes in system behavior are controlled by activated transitions. In our approach we extended the classical petri-net concept by so-called *Active Memory Guards* (AMG) which utilize event listeners to connect the model to instances of Active Memory XML Servers. An input arc can only be satisfied after, firstly, the attached place contains a sufficient amount of tokens thus enabling the memory event listener, and secondly, an occurrence of the specified memory event in an active memory instance. If both steps are completed, the input arc as a whole is satisfied and the attached transition is enabled. This concept couples the execution of the specified high-level petri-net model to the overall state. AMGs are context dependent in a sense that they are activated as soon as the place condition of its arc is satisfied.

The realization of the active memory petri-net engine allows a formal and declarative specification of net structure and active memory guards as well as the attached actions in an application of the PNML document format [WK03]. Thus, petri-net coordination models can be extended by new places and tran-

[2] The so-called Intrinsic Memory Processes are explained in greater detail in [WHBS04].

sitions online. As soon as a PNML model is updated in an active memory server, the instantiated petri-net execution engine is reconfigured.

Fig. 4 shows on a conceptual level how a petri-net control process is coupled to the overall system. As soon as a transition fires, a sequence of actions is executed. The set of possible actions which can be attached to a transition can be any number of XCF calls, basic actions on a memory server or local calls to methods of classes that are derived from a basic action interface. Instances of those actions are specified in the PNML model and can be configured with XML parameters.

2.5 Introspection and Compacting

Introspection in general is a very helpful feature for software integration because it helps a lot in debugging and monitoring a running distributed system. In XCF, the basic communication mechanisms as well as the Active Memory XML Server are reflexive in that all kinds of data and process configurations are represented in XML data that can be analyzed and changed on-the-fly. Furthermore, the type of actions and their context can be inspected during runtime. The framework architecture allows to intercept all messages within a running system or to intercept individual communication between specific components. In XCF, the features for introspection are realized with an implementation of the interceptor pattern [SSRB00].

An exemplary application that benefits from the reflection XML itself provides, is the so-called *StreamSimulator* which is able to simulate XCF Publisher components by replaying XML data from previously recorded files or from the contents of an Active Memory XML Server. In order to *peek* inside an active memory server, a generic visualization toolkit, the Memory Content Introspector (MCI), has been developed (see Fig. 6). The MCI is realized as a plugin-framework written in Java where each of the plugins realizes a specific visualization method.

A more complex application of the introspective features is the generic compacting filter for XML streams [LWW05] which we developed to reduce waste of system resources. An example for this is object recognition, where usually parts of the transmitted data may be false when the robot looks around, other data may be redundant, e.g. when the robot gazes at a static scene. In this situation object recognition will work well but (at least the implementation we had at hand) sends nearly identical object recognition results at a high frequency. Both, redundant and obviously false recognition results consume resources of the system in terms of network bandwidth and overall system performance. Therefore, a generic framework component based on reflection was developed that is able to compact processed information and detect relevant changes in a stream of XML information. Thus, only the relevant data is forwarded by the compacting filter to subsequent processing modules. This allows us to integrate existing modules even if their temporal

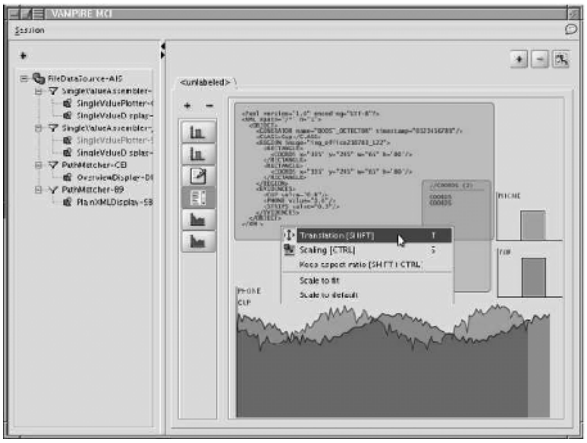


Fig. 6. Screenshot of the Memory Content Introspector.

behavior does not fit the input demands of other modules in the integrated system processing their data.

In the following we show how these foundations are used to realize the integration of different modules for human-robot interaction and which lessons we learned so far by applying our approach during the development of a complex interactive robot.

3 Developing an Interactive Robot

Through integrating a large number of components using the tools presented above we realized a personal robot that has already quite impressive interaction capabilities [LHW05]. The mobile robot BIRON depicted in Fig. 7 is able to pay attention to different persons and engage in a one-to-one interaction with one user if this user greets the robot by saying “Hello Robot”. From this point on, the robot focuses on this communication partner and engages in a dialog with him. The user can control the robot’s movement behavior by giving commands. For example, the command “Follow me” results in the robot following the human around. More importantly, the user can teach new objects to the robot by pointing at them while giving additional information. For example, giving an instruction like “This <gesture> is my blue cup” enables the robot to focus its attention on the referenced object and acquire an image of it for later recognition. Components for localization and navigation enable the user to teach the robot places and locations as well as to enable the robot to autonomously go to verbally specified locations (e.g., ‘Go to the kitchen’).

The sketched functionality is achieved by integrating modules for robot control, person tracking, person attention, speech recognition, speech under-

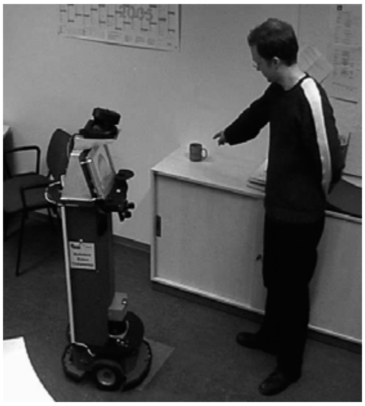


Fig. 7. A user teaching an object to the mobile robot BIRON.

standing, dialog, gesture recognition, object recognition, object attention and an execution supervisor [FKH05]. The module integration task directly benefits from the fact that XCF is very easy to use and the applied concepts are highly standards-based, giving interdisciplinary researchers a quick start. Furthermore, a central system view of the active robot modules and the exchanged data utilizing the active system introspection supports debugging and evaluation of the running system. Loose coupling of modules and the declarative style of accessing system data paid off in ease of modification and the ability to let the system architecture evolve over time as we extended it with new modules and data types. With several researchers in the European project COGNIRON [Cog05] – The Cognitive Robot Companion – contributing to this system and extending the functionality of their respective modules frequently, the use of XML schemas for data type verification proved to be useful for identifying modules providing erroneous data. The active memory enables sharing of knowledge acquired during the robot's interaction with a human instructor. Through event subscription at run-time, different types of newly acquired knowledge result in different modules being notified and, consequently, only processing relevant to the new data is performed.

With respect to the performance of the overall system, the distribution and coordination of modules has resulted in a high reactivity allowing for more natural human-robot interaction. After the end of a user instruction the system produces a response within 400-700ms [FKH05]. Most of this delay is caused by the speech recognition process (200-500ms), whereas the processing by all other modules and the time needed for communication is only around 200 ms. Transferring a message from one module to another module takes only a few milliseconds and shows that the advantage of increased processing power in a distributed system setup for a personal robot exceeds the cost for module communication.

It should be noted here, however, that the proposed communication framework is not intended for integration of low-level robot control functionalities like, e.g., obstacle avoidance. For this task other frameworks described in this Chapter like, e.g., Player or Marie are possibly better suited. While these are especially focused on the needs of robotics researchers, the XCF SDK is especially designed for supporting the development and integration of high-level functionalities like speech processing and gesture recognition.

4 Lessons Learned

Using XCF most of the data flow between all modules in the architecture is event-based and every message is coded in XML. As both, data structures and communication channels, are *name-based* the different modules can easily be replaced by others. It also eases modification concerning the data structures used for communication, as XML allows database-like exchange of information. Experience in different European research projects showed that the use of XML helped in defining data types which are suitable for every involved project partner. XML objects as native data types allow for a flexible schema evolution and separation of concerns within a system architecture. Additionally, generic software modules can much easier be realized within this concept.

In the practical use of schemas during prototyping, exceptions were encountered frequently. This was often due to misunderstandings between the involved developers about data type definitions, but occasionally also serious implementation errors were discovered. For example, a module that provided laser range data threw an exception immediately after the start stating that the XML laser data was not valid. Without schema checking being active, the module started and seemed to operate correctly, the transmitted XML data looked correct, and even the communication worked fine. Only after a more detailed analysis it was found that *at startup* the very first data packet obtained from the laser range finder was corrupt and this caused the exception.

Concerning the goals mentioned in the beginning of this Chapter, we think that *usability* of the XCF toolkit is high. This emerges from two design decisions: On the one hand the component developer only has to deal with a small number of classes in a simple API and on the other hand many technologies used are standards-based. Using XML technologies throughout the whole framework, we meet the *flexibility* requirement. This results in changeability, easier adaptation and integration of modules. Additionally, openness of distributed memory instances allows developers to retrieve information in a standard fashion using declarative queries.

Low coupling is reached through combination of active memory instances and XCF. The memory instances itself decouple the memory processes and serve as an information mediator while the XCF framework provides location and access transparency for components. This leads to easy exchange of components and high robustness against component failure. Both techniques enable *rapid prototyping*.

Debugging and evaluation is supported by simulation of components using XCF for replaying recorded memory data with a so called module simulator. When the memory content has time information associated whole architectural layers can be replaced by simulation tools as if they were online available. Development and evaluation of different algorithms or system configurations on comparable data has been much easier with this feature. For the task of system integration, parts of the system could be tested by simulating the communication with other components. The introspection facilities turned out to be highly valuable for monitoring the overall system state and the correct processing of data at the system integration level. The central logging of information made the task of system integration much easier as it enabled the monitoring of process chains typical for complex interactive robots.

Finally it should be mentioned that all the lessons learned could only be learned by actually having succeeded in motivating different researchers to combine their research systems for isolated interaction aspects in a single integrated robot prototype. We believe that XCF has played a crucial role in this outcome, as it has made it comparatively easy for the involved people to extend their modules with communication facilities.

5 Conclusion

While this Chapter outlined the concepts of the XCF SDK, the interested reader is referred to the webpage [Wr05] for examples and implementation level documentation. Concluding our experiences, the successful realization of a personal robot prototype clearly demonstrates that the tools described in Sect. 2 are usable by interdisciplinary researchers. The proposed framework has shown to adequately support an evolutionary development process and the tools are well suited to be used in such large-scale research projects. Nevertheless, integration is still a challenging issue as it is often underrepresented in research projects but is unavoidable especially if multi-modal interactive intelligent systems are to be developed. The presented data- and event-driven integration method enables the re-use of modules in different application scenarios and we, therefore, consider this coordination scheme to pave the way for incremental development of more complex robot prototypes in the future.

References

[BMRS96] F. Buschmann, R. Meunier, H. Rohnert, and P. Sommerlad, *Pattern-oriented software architecture*, vol. 1: A System of Patterns, John Wiley & Sons Ltd., 1996.

[CD99] J. Clark and S. DeRose, *XML Path Language*, Tech. Report REC-xpath-19991116, W3C, 1999.

[Cog05] Cogniron Consortium, *COGNIRON – The Cognitive Robot Companion*, 2005, `http://www.cogniron.org`.

[CS00] È. Coste-Manière and R. G. Simmons, *Architecture, the Backbone of Robotic Systems*, Proc. IEEE Int. Conf. on Robotics and Automation (San Franciso, CA), vol. 1, 2000, pp. 67–72.

[DBX03] *Berkely DB XML, Sleepycat Software*, 2003, `http://www.sleepycat.com/products/xml.shtml`.

[FKH05] J. Fritsch, M. Kleinehagenbrock, A. Haasch, S. Wrede, and G. Sagerer, *A flexible infrastructure for the development of a robot companion with extensible HRI-capabilities*, Proc. IEEE Int. Conf. on Robotics and Automation (Barcelona, Spain), April 2005, pp. 3419–3425.

[KAU04] P. Kiatisevi, V. Ampornaramveth, and H. Ueno, *A Distributed Architecture for Knowledge-Based Interactive Robots*, Proc. Int. Conf. on Information Technology for Application (ICITA) (Harbin, China), 2004, pp. 256–261.

[LHW05] S. Li, A. Haasch, B. Wrede, J. Fritsch, and G. Sagerer, *Human-style interaction with a robot for cooperative learning of scene objects*, Proc. Int. Conf. on Multimodal Interfaces (Trento, Italy), ACM Press, 2005, pp. 151–158.

[LWW05] I. Lütkebohle, S. Wrede, and S. Wachsmuth, *Unsupervised Filtering of XML Streams for System Integration*, International Workshop on Pattern Recognition in Information Systems, 2005, Poster.

[Pet81] J. L. Peterson, *Petri net theory and the modeling of systems*, Prentice Hall, Inc., Englewood Cliffs, Massachusetts,, 1981.

[Ros95] J. K. Rosenblatt, *DAMN: A Distributed Architecture for Mobile Navigation*, Proc. AAAI Spring Symp. on Lessons Learned from Implemented Software Architectures for Physical Agents (Stanford, CA), AAAI/MIT Press, 1995, pp. 167–178.

[Wr05] S. Wrede, *The XML enabled Communication Framework SDK*, 2005, Software and documentation available at `http://xcf.sf.net/`.

[SSRB00] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-oriented software architecture*, vol. 2: Patterns for Concurrent and Networked Objects, John Wiley & Sons Ltd., 2000.

[Vam05] Vampire Consortium, *VAMPIRE – Visual Active Memory Processes for Interactive Retrieval*, 2005, `http://www.vampire-project.org`.

[WFBS04] S. Wrede, J. Fritsch, C. Bauckhage, and G. Sagerer, *An XML Based Framework for Cognitive Vision Architectures*, Proc. Int. Conf. on Pattern Recognition, vol. 1, 2004, pp. 757–760.

[WHBS04] S. Wrede, M. Hanheide, C. Bauckhage, and G. Sagerer, *An active memory as a model for information fusion*, Proc. 7th Int. Conf. on Information Fusion, 2004, pp. 198–205.

[WK03] M. Weber and E. Kindler, *The petri net markup language*, Petri Net Technology for Communication Based Systems., LNCS 2472, Springer-Verlag, 2003.

[Wor05] World Wide Web Consortium, *XML-binary Optimized Packaging, W3C Recommendation 25 January 2005*, 2005, `http://www.w3.org/TR/2005/REC-xop10-20050125/`.

[WWHB05] S. Wachsmuth, S. Wrede, M. Hanheide, and C. Bauckhage, *An Active Memory Model for Cognitive Computer Vision Systems*, Künstliche Intelligenz **19** (2005), no. 2, 25–31.

[Zer05] ZeroC Inc., *The Internet Communications Engine, ZeroC Inc.*, 2005, `http://www.zeroc.com/ice.html`.