
Building Multimodal Dialogue Applications: System Integration in SmartKom

Gerd Herzog and Alassane Ndiaye

DFKI GmbH, Stuhlsatzenhausweg 3, D-66123 Saarbrücken
{herzog,ndiaye}@dfki.de

Summary. In this contribution, we will report on the experience gained in building large-scale research prototypes of fully integrated multimodal dialogue systems in the context of the SmartKom project. The development of such systems requires a flexible software architecture and adequate software support to cope with the challenge of system integration. A practical result of our experimental work is an advanced integration platform that enables flexible re-use and extension of existing software modules and is able to deal with a heterogeneous software environment. Starting from the foundations of our general framework, an overview of the SmartKom testbed will be given and we will describe also the practical organisation of the development process within the project.

1 Background

The realisation of fully-fledged prototype systems for various application scenarios constituted an integral part of the research and development activities within the SmartKom project [16, 17]. Building such large-scale multimodal dialogue applications poses a significant technical challenge and requires a principled approach for successful system integration.

Figure 1 provides an overview concerning the technical components of the SmartKom demonstrator system which covers all application scenarios, including SmartKom Public, Home, and Mobile as well as the English system. The instrumented vehicle, which is not permanently available for demonstrations, can also be connected to this demonstrator setup. Alternatively, standard audio components and a separate display can be employed to simulate the in-car equipment. Furthermore, a GPS simulation can be used for SmartKom Mobile to demonstrate incremental route guidance. The overall system incorporates more than 40 software components from 11 different project partners. Customised SmartKom installations exist at all partner sites. For example, 2 laptops and a handheld computer are sufficient for a compact instantiation of SmartKom Mobile.

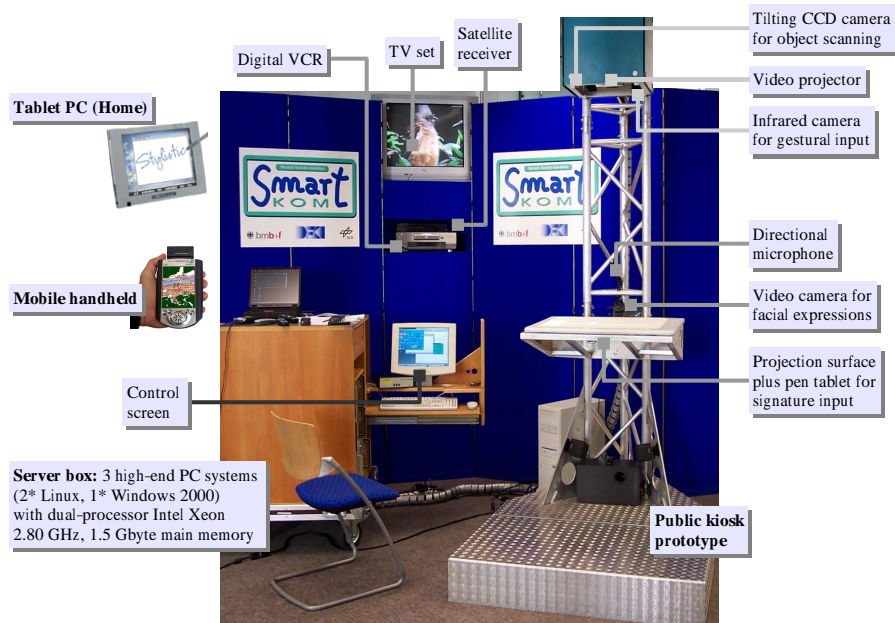


Fig. 1. SmartKom reference installation for demonstrations

The difficulty of system integration is high and often underestimated. The distributed nature of a large-scale research activity like SmartKom imposes severe constraints on the development of an integrated system. This kind of joint effort is characterised by a lot of variance in code development across teams. Each workgroup is accustomed to its preferred software environment and development practices. Given the need for optimal use of available project resources, it is also not feasible to start an implementation from scratch. The resulting heterogeneity of contributions from different project partners impedes the realisation of a monolithic system. Thus adequate technical and organisational means are required to assemble heterogeneous components into an integrated and fully operational system.

The following section presents some basic considerations that govern our technical approach for system integration. Section 3 then provides a compact overview of the SmartKom testbed, our specific software integration platform that has been employed to realise the multimodal dialogue system prototypes. In addition to technical issues, organisational aspects are of particular importance for a successful research and development project. Section 4 describes how the development and integration process has been organised within the SmartKom project.

2 A Framework for Software Integration

Aiming at the construction of advanced intelligent user interfaces typical aspects of large-scale system development have to be considered. The following key factors influence the technical approach to be taken:

- Strong need to reuse and extend existing software;
- Pressure to build initial prototypes rapidly;
- Significant number of distinct components from different sources;
- Heterogeneous software environment, including various programming languages and different operating systems;
- High variability of target configurations for the integrated dialogue system applications.

A distributed system constitutes the natural choice to realize an open, flexible and scalable software architecture, able to integrate heterogeneous software modules implemented in diverse programming languages and running on different operating systems. A key point of our specific integration framework is the idea of having a component-based system, composed out of executable modules. Our approach proposes a modularisation of the dialogue system into distinct and independent software modules to allow maximum decoupling. A main advantage of such a component architecture [14] is the fact that component integration and deployment are independent of the development life cycle, and there is no need to recompile or relink the entire application when updating with a new implementation of a component.

The kind of component-based software engineering proposed here requires an integration platform that provides the necessary software infrastructure to assemble the large-grained components into a multimodal dialogue system. Such an integration platform can be characterised as a software development kit and consists of the following main parts:

- A run-time platform for the execution of a distributed dialogue application, including also means for configuring and deploying the individual parts of the complete multimodal dialogue system;
- Application programming interfaces (API) for linking into the run-time environment and for interprocess communication;
- Tools and utilities to support the whole development process, including also installation and software distribution.

In general, different options are available for the interconnection of individual components [2, 6, 9]. Distributed processing with remote procedure calls or remote method invocation follows the client-server paradigm and using these techniques, each component has to declare and implement a specific API to make its encapsulated functionality transparently available for other system modules. Asynchronous message-based communication, however, better supports the need for scalability, flexibility, and decoupling.

Two main schemes of message-oriented middleware can be distinguished. Basic *point-to-point* messaging employs unicast routing and realises the notion of a direct connection between message sender and a known receiver. The more general *publish/subscribe* approach is based on multicast addressing. Instead of addressing one or several receivers directly, the sender publishes a notification on a named message queue, so that the message can be forwarded to a list of subscribers. This kind of distributed event notification makes the communication framework very flexible as it focusses on the data to be exchanged and it decouples data producers and data consumers. The well-known concept of a blackboard architecture [7] and distributed shared memory approaches that are based on the notion of tuple spaces [4] follow similar ideas.

In order to ensure syntactic and semantic interoperability definite contracts among components are required. Using message-based communication, the interface of a specific component can be characterised by the admissible message contents that it is able to process and to produce. XML, the so-called *extensible markup language*, has meanwhile evolved into a standard tool for the flexible definition of application-specific data formats for information exchange. Thus we propose to operationalise interface specifications in the form of an XML language. XML-based languages define an external notation for the representation of structured data and simplify the interchange of complex data between the separate components of an integrated multimodal dialogue system.

For the technical realisation of an integration platform, we postulate basic design principles that form the conceptual foundation of our integration framework:

- Asynchronous message-passing is more flexible than component-specific remote APIs and better fits parallel processing in a distributed system.
- Blackboard-like publish/subscribe messaging is a generalisation of point-to-point communication which supports decoupling of components and allows better modularisation of data interfaces.
- Well designed XML languages support the definition of declarative data models for information interchange and semantically rich interface specifications.
- The proposed framework does not rely on centralised control. Instead of that, individual components take care if coordination is needed. Required synchronisation information is modelled explicitly within data interfaces.
- There should be no hidden interaction between components. All data has to be exchanged transparently via the provided communication system to ensure traceability of processing behaviour.

3 The SmartKom Testbed

Taking into account the design aspects outlined in the previous section, it becomes obvious that available off-the-shelf middleware is not enough to pro-

vide a flexible architecture framework and a powerful integration platform for large-scale dialogue systems. These considerations motivated our investment into the development of a suitable system integration platform.

SmartKom relies on the so-called *Multiplatform Testbed* [12], a software solution that originates from our practical work in various research projects. The Testbed, which is built on top of open source software [18], provides a fully-fledged integration platform for dialogue systems and has been improved continuously during the last several years [3, 13, 15].

3.1 Middleware Solution

Our specific realisation of an integration platform follows the architecture framework and the design principles described in the previous section. Its main characteristics are:

- Component architecture
 - Modularisation into distinct and independent software modules to allow maximum decoupling;
 - Encapsulation of heterogeneous, large-grained software components into a uniform module structure.
- Event-based architectural style
 - Very flexible publish/subscribe messaging;
 - Implicit invocation (event-driven) to support decoupling of components;
 - Highly adaptable content format specifications for each data pool, i.e. named message queue.

Instead of using custom programming interfaces, the interaction between distributed components within the Testbed framework is based on the exchange of structured data through messages. The SmartKom Testbed includes a message-oriented middleware implementation that is based on PVM, the *parallel virtual machine* software described in [10]. In order to provide publish/subscribe messaging with data pools on top of PVM, a further software layer called PCA (*pool communication architecture*) had to be added [15].

Compared with commercial grade middleware products our low cost solution does not consider issues like fault tolerance, load balancing or replication. It does, however, provide a scalable and efficient platform for the kind of real-time interaction needed within a multimodal dialogue system. The PVM/PCA-based messaging system is able to transfer arbitrary data contents and provides excellent performance characteristics. Within SmartKom Public, for example, it is even possible to perform a telephone conversation. Message throughput on standard PCs with Intel Pentium III 500 MHz CPU is off-hand sufficient to establish a reliable bi-directional audio connection, where uncompressed audio data are being transferred as XML messages in real-time.

The so-called *module manager* provides a thin API layer for component developers with language bindings for the programming languages that are used to implement specific dialog components. It comprises the operations required to access the communication system and to realise an elementary component protocol needed for basic coordination of all participating distributed components. The module manager implementation of the SmartKom Testbed includes in particular APIs for C, C++, Java, and Prolog.

3.2 Technical Modules for System Execution

In addition to the functional components of the dialogue system, the runtime environment includes also special Testbed modules in support of system operation.

The *Testbed manager* component is responsible for system initialisation and activates all distributed components pertaining to a given dialogue system configuration. It also cooperates with the functional modules to carry out the elementary component protocol, which is needed for proper system start-up, controlled termination of processes and restart of single components, or a complete soft reset of the entire dialog system. A state-transition network forms the basis for this protocol, which is also directly supported by the API functions of the module manager interface. As illustrated in Fig. 2, only a few basic component states need to be distinguished.

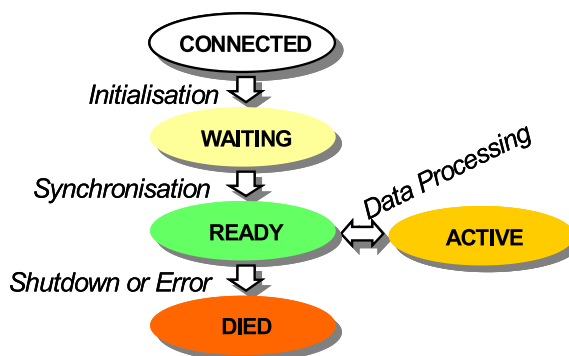


Fig. 2. Basic component states for module coordination

The Testbed manager supports flexible runtime configuration of the application system. Module configurations can be declared using an XML-based specification. The modular concept allows startup of individual components or sub-systems for debugging purposes, e.g. to test a critical path from speech or gesture input to speech and graphic output.

The *Testbed control GUI* shown in Fig. 3 constitutes a separate component which provides a graphical user interface for the administration of a running

system. The adaptable Testbed GUI provides a simplified view of the system architecture and offers the necessary means to monitor system activity, to interact with the Testbed manager, and to manually modify configuration settings of individual components while testing the integrated system.

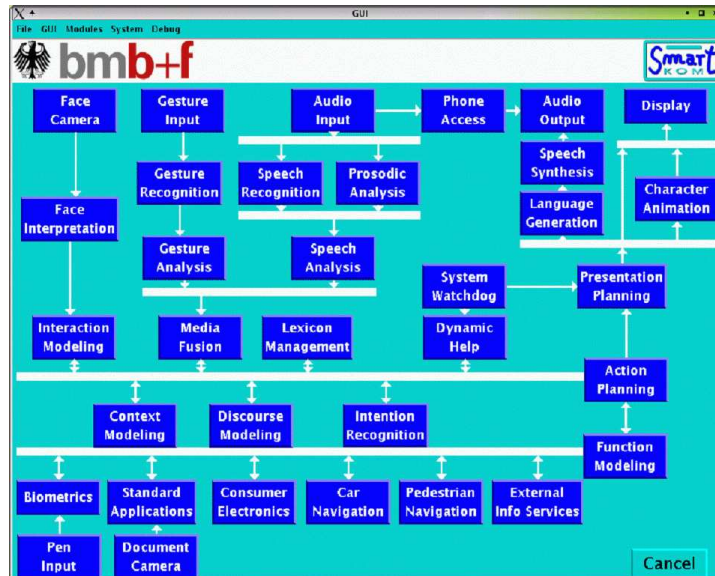


Fig. 3. Graphical user interface for Testbed administration

Both Testbed modules and the functional components interact using the underlying communication system. Specific data pools are employed to exchange Testbed-related control data vs. application data of the multimodal dialogue system. From the point of view of the component developer, the internal system pools are not directly visible as they are hidden inside the module manager API implementation.

The SmartKom Testbed provides automatic logging of module output messages and all communicated data. A further Testbed module, the logging component, is being employed to save a complete protocol of all exchanged messages—including system pools—for later inspection.

3.3 M3L Support

The SmartKom system employs M3L, the *Multimodal Markup Language*, as external data format for information exchange between functional components. M3L, an XML-based language, has been developed within the SmartKom project to cover all data interfaces inside this complex multimodal dialogue system.

Parts of the language specification are generated automatically from an underlying conceptual taxonomy, that provides the foundation for the representation of domain knowledge inside the dialogue system. The offline tool described in [11] transforms an ontology written in OIL format (see [8]) into an M3L compatible XML Schema definition. Our tool simplifies updates of the M3L specification as the ontology evolves and ensures a consistent mapping from structural and semantic knowledge encoded in the ontology to directly interpretable communication interfaces.

From the point of view of component development, XML offers various techniques for the processing of transferred content structures. The standard DOM API makes the M3L data available as a generic tree structure—the *document object model*—in terms of elements and attributes, whereas SAX (*simple API for XML*) provides an event-based API for XML parsers.

Another interesting option is to employ XSLT stylesheets to flexibly transform between the external XML format used for communication and a given internal markup language of the specific component. The use of XSLT makes it easier to adapt a component to interface modifications and simplifies its re-use in another dialogue system.

Instead of working on basic XML structures like elements and attributes, XML data binding can be used for a direct mapping between program internal data structures and application-specific M3L markup. In this approach, the M3L language specification is exploited to automatically generate a corresponding object model in a given programming language.

All the techniques described here have been applied in different implementations of the various functional components of the SmartKom system. In addition, a specific M3L API has been developed and included into the SmartKom Testbed. This M3L API offers for all target platforms a lightweight programming interface to simplify the processing of XML structures within the implementation of a component. It incorporates a standard XML parser and uses the DOM interface to implement the M3L-specific API layer.

Furthermore, the SmartKom Testbed also provides a generic, XML-enabled toolbox and tailored XSLT stylesheets for the offline and online inspection of M3L data.

3.4 Additional Testbed Tools

A rich set of utilities can be used to manipulate log data easily, e.g. to select and identify relevant subsets from the complete data store resulting from a system run.

Flexible replay of selected pool data provides a simple, yet elegant and powerful mechanism for the simulation of small or complex parts of the dialog system in order to test and debug components during the development process. Using a simple tool, a system tester is also able to generate trace messages on the fly to inject annotations into the message log.

Another important development tool is a generic, XML-enabled data viewer for the online and offline inspection of pool data. Figure 4 presents an example of a word hypothesis graph, corresponding to the natural language part of a multimodal user input. Such a graphical visualisation of speech



Fig. 4. Word hypothesis graph for “I would like to know more about this”.

recognition results during system execution aids the system integrator while testing the dialogue system.

The flexible data viewer toolbox can also be used in offline mode. A practical example is given in Fig. 5, which shows a comprehensible display format that documents an interaction with the Smartkom system. This kind of dialogue protocol provides a compact summary of important processing results and is generated fully automatically from the message log. The individual entries consist of the specific message number followed by the summarised content of selected data pools. Each item is prepended with the corresponding message number in order to enable more detailed inspection of relevant data if potential processing errors need to be resolved.

Further offline tools include a standardised build and installation procedure for components as well as utilities for the preparation of software distributions and incremental updates during system integration.

4 System Integration Process

In addition to the software infrastructure, the practical organisation of the project constitutes a key factor for the successful realisation of an integrated multimodal dialogue system. Distributed teams and heterogeneous software environments within a large research project add complexity that must be resolved. In order to cope with system integration challenges it is beneficial to clearly distinguish two major roles in the development process. A system integrator takes care of the overall software infrastructure and assembles executable components into an integrated application system, whereas a module developer should concentrate on a specific functional component.

For the SmartKom project, a dedicated system integration group composed of professional software engineers has been established to ensure that the resulting software is robust and maintainable, that an adequate architectural framework and testbed are provided to the participating researchers, that software engineering standards are respected, and that modules developed in different programming languages by a distributed team fit together

...


02429 User comment: SmartKom displays the TV program for the evening

03699 #PAUSE# was kommt Sonntag Abend im Fernsehen #PAUSE#

03779 was kommt Sonntag Abend im Fernsehen.. (1.0)
was kommt Sonntag Abend im Fernsehen.. (0.8)

03949 informationSearch InformationType=tvProgram weekday='sunday' daytime='evening'
from='2003-06-08T18:00:00' to='2003-06-08T23:59:59'

04408 hier sehen Sie die gewünschten Sendungen



05182


05254 User comment: SmartKom provides information about the movie *Evolver*

06789 #PAUSE# ich will mehr darüber wissen #PAUSE#

06821 ich will mehr darüber wissen.. (1.0)

06989 informationSearch InformationType=broadcast at='2003-06-08T22:10:00'
at='2003-06-08T23:50:00' avType='scienceFiction' title='Evolver' name='Kabel1'

07278 ich habe diese Informationen zu Evolver



07809

...

Fig. 5. Excerpt from an automatically generated dialogue protocol

properly. These aspects are too important to leave them to the individual scientists who regard them as a side issue because they have to focus on their own research topics. An important achievement of the Smartkom project is the consistent integration of a very large number of components created by diverse groups of researchers from disparate disciplines.

Given the underlying architecture framework, the approach taken in SmartKom bears many similarities to component-based software engineering, where the notion of building an application by writing code has been replaced with building a system by assembling and integrating supplied software components [1]. In contrast to conventional software development, where system integration is often the tail end of an implementation effort, component integration is the centrepiece of the approach.

Figure 6 provides a sketch of our development process, which is integration-centric as opposed to development-centric. In this kind of view, implementation has given way to integration as the focus of system construction.

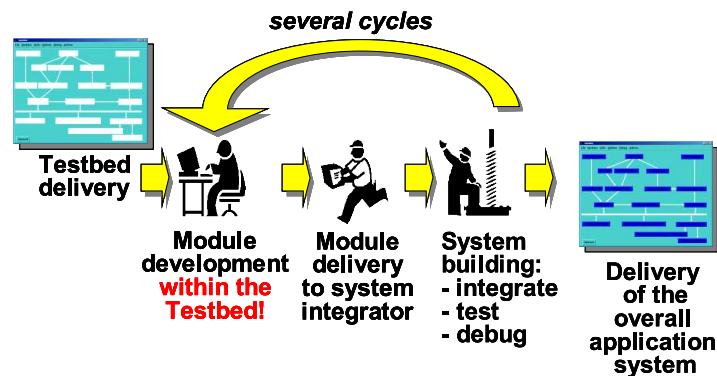


Fig. 6. Stages of system integration

After the design phase, which leads to a specification of the concrete component architecture and functional interfaces, the realisation starts with the provision of a tailored Testbed software package. The SmartKom Testbed follows the *breadboard paradigm*, which provides an initial instantiation of the overall architecture using placeholders that represent the functional components [3, 15]. During system integration, the dummy components are replaced successively by more substantial modules with increased functionality, i.e. components are like black boxes which can be plugged into a breadboard. Non-existent modules can be simulated through the replay of appropriate test data. This technique enables rapid prototyping given that even at an early stage an environment is available for testing and evaluating individual modules in the context of the entire application.

The one constant aspect of component-based software development is change. Constituent components are constantly evolving, leading to an iterative process where the different phases have to be repeated several times until the multimodal dialogue application can be finalised. This includes also the customisation of the software infrastructure, which is an integral element of a project life-cycle.

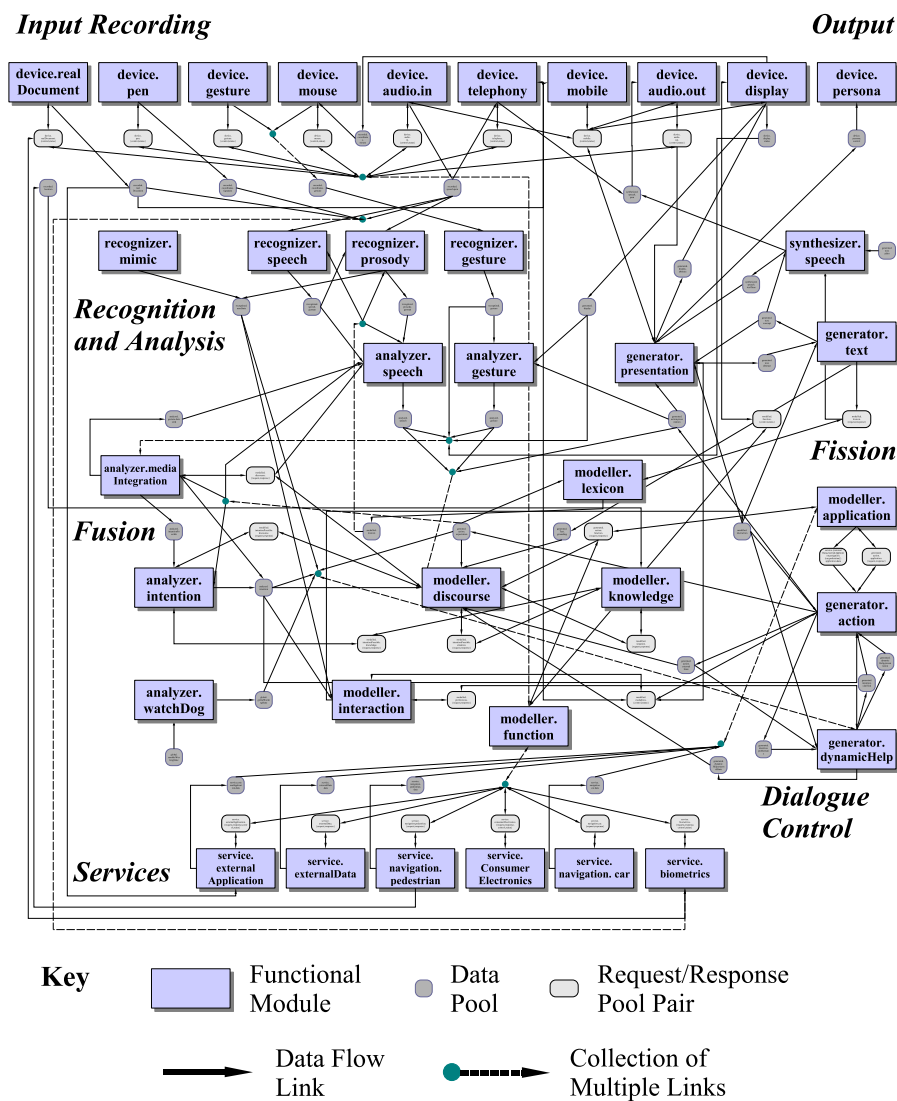


Fig. 7. Detailed software architecture of the SmartKom system

Stepwise improvement and implementation of the design of architecture details and interfaces necessitate an intensive discussion process. It has to include all participants involved in the realisation of system components in order to reach a common understanding of the intended system behaviour. The moderation and coordination of these activities constituted an important task of the system integration group in SmartKom.

The functional architecture of the multimodal dialogue system provides a basis for modularisation into distinct software components. Significant parts of the initial system architecture were already pre-determined by the project structure. Functional components are mapped to modules, realised as independent software processes, to obtain a specific software architecture. Figure 7 depicts the components and communication links inside the SmartKom system, which includes about 40 functional modules and 100 data pools for information exchange.

The specification of the content format for each data pool defines the common language that the dialogue system components use to interoperate. The careful design of information flow and accurate specification of content formats act as essential elements of our approach. We operationalise interface specifications in the form of an XML language. Our experience with M3L shows, that the design of such an XML language for the external representation of complex data is not a simple task. All design decisions have to be made carefully. For example, it is better to minimise the use of attributes. They are limited to unstructured data and may occur at most once within a single element. Preferring elements over attributes better supports the evolution of a specification since the content model of an element can easily be redefined to be structured and the maximum number of occurrences can simply be increased to more than one. A further principle for a well-designed XML language requires that the element structure reflects all details of the inherent structure of the represented data, i.e., textual content for an element should be restricted to well-defined elementary types. Another important guideline is to apply strict naming rules so that it becomes easier to grasp the intended meaning of specific XML structures.

The latest version of M3L consists of more than 1,000 different XML elements. In order to make the specification process manageable and to provide a thematic organisation, the M3L language definition has been decomposed into about 40 schema specifications. This kind of modularisation allowed to distribute responsibility for the different parts of the specification, assigning each XML schema to one selected researcher with specific expertise in that area.

5 Concluding Remarks

This contribution has presented a discussion on system integration in building multimodal dialogue applications with an emphasis on the SmartKom integration platform. It has shown the analogy of our approach to component-based software engineering.

The capabilities of the Multiplatform software framework are being enhanced on a continuing basis since the first design in Verbmobil, the predecessor of the SmartKom project. The SmartKom Testbed provides a powerful environment and advanced new tools, like for example the added support for

XML-based data representations. Multiplatform has been made available as open source software and is currently being used at several industrial and academic sites.

The paper also gave a look into the underlying system integration process and underlines the role of non-technical factors for effective inter- and intra-institutional collaboration.

Acknowledgement

First of all, we would like to thank our former colleagues in the system integration group. Heinz Kirchmann, Andreas Klüter, Hans-Jörg Kroner, Stefan Merten, Martin Neumann, and Rainer Peukert all contributed as well to the work described here.

Given the central position of our team within the project structure, there have been intensive contacts with nearly every project participant. We enjoyed this joint work and gratefully acknowledge the very successful collaboration in the SmartKom consortium.

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, Boston, 2nd edition, 2003.
2. P. A. Bernstein. Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39(2):86–98, 1989.
3. T. Bub and J. Schwinn. The Verbmobil Prototype System—A Software Engineering Perspective. *Natural Language Engineering*, 5(1):95–112, 1999.
4. N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, 1989.
5. H. Cunningham and J. Patrick, editors. *HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS)*, Edmonton, Canada, 2003. Association for Computational Linguistics.
6. W. Emmerich. Software Engineering and Middleware: A Roadmap. In *Proc. of the Conf. on the Future of Software Engineering*, pages 117–129. ACM Press, 2000.
7. L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Surveys*, 12(2):213–253, 1980.
8. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
9. K. Geihs. Middleware Challenges Ahead. *IEEE Computer*, 34(6):24–31, 2001.
10. A. Geist, A. Beguelin, J. Dongorra, W. Jiang, R. Manchek, and V. Sunderman. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, 1994.

11. I. Gurevych, S. Merten, and R. Porzel. Automatic Creation of Interface Specifications from Ontologies. In Cunningham and Patrick [5], pages 59–66.
12. G. Herzog, H. Kirchmann, S. Merten, A. Ndiaye, P. Poller, and T. Becker. MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems. In Cunningham and Patrick [5], pages 75–82.
13. G. Herzog, A. Ndiaye, S. Merten, H. Kirchmann, T. Becker, and P. Poller. Large-scale Software Integration for Spoken Language and Multimodal Dialog Systems. *Natural Language Engineering*, 10, 2004. Special issue on Software Architecture for Language Engineering.
14. J. Hopkins. Component Primer. *Communications of the ACM*, 43(10):27–30, 2000.
15. A. Klüter, A. Ndiaye, and H. Kirchmann. Verbmobil From a Software Engineering Point of View: System Design and Software Integration. pages 635–658. Springer, Berlin, 2000.
16. N. Reithinger, G. Herzog, and A. Ndiaye. Situated Multimodal Interaction in SmartKom. *Computers & Graphics*, 27(6):899–903, 2003.
17. W. Wahlster. SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell. In R. Krahl and D. Günther, editors, *Proc. of the Human Computer Interaction Status Conference 2003*, pages 47–62, Berlin, Germany, 2003. DLR.
18. M.-W. Wu and Y.-D. Lin. Open Source Software Development: An Overview. *IEEE Computer*, 34(6):33–38, 2001.