Large-Scale Software Integration for Spoken Language and Multimodal Dialog Systems

Gerd Herzog, Alassane Ndiaye, Stefan Merten,

Heinz Kirchmann, Tilman Becker, Peter Poller

German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, D–66123 Saarbrücken, Germany
{herzog,ndiaye,merten,kirchman,becker,poller}@dfki.de

(*Author draft for publication in Natural Language Engineering 10(3/4), pp. 283–305*)

Abstract

The development of large-scale dialog systems requires a flexible architecture model and adequate software support to cope with the challenge of system integration. This contribution[1] presents a general framework for building integrated natural-language and multimodal dialog systems. Our approach relies on a distributed component model that enables flexible re-use and extension of existing software modules and is able to deal with a heterogeneous software environment. A practical result of our research is the development of a sophisticated integration platform, called MULTIPLATFORM, which is based on the proposed framework. This MULTIPLATFORM testbed has been used in various large and mid-size research projects to develop integrated system prototypes.

1 Introduction

One central element of research in the field of intelligent user interfaces (Maybury and Wahlster 1998) is the construction of advanced natural language and multimodal dialog systems that demonstrate the high potential for more natural and much more powerful human-computer interaction. Although language technology already found its way into fundamental software products—as exemplified by the Microsoft Speech SDK (*software development kit*) for Windows and the Java Speech API (*application programming interface*)—the development of novel research prototypes still constitutes a demanding challenge (Leidner 2003). State-of-the-art dialog systems combine practical results from various research areas and tend to be rather complex software systems which can not simply be realized as a monolithic desktop computer application. More elaborate software designs are required in order to assemble heterogeneous components into an integrated and fully operational system (Bub and Schwinn 1999).

[1] Our current work in the context of the SmartKom project has been funded by the German Federal Ministry for Education and Research (BMBF) under grant 01 IL 905 K7.

A typical research project involves several, sometimes even more than 20 work groups from different industrial and scientific partners, leading to a broad spectrum of practices and preferences that govern the development of software components. In particular, a common software platform for the construction of an integrated dialog system often needs to support different programming languages and operating systems so that already existing software can be re-used and extended. Taking into account the potential costs it is usually not feasible to start an implementation from scratch. Another important aspect is the use of rapid prototyping for accelerated progress which leads to frequent changes in design and implementation as the project unfolds.

Playing the role of a dedicated system integration group during the last twelve years, we have been involved in various research projects aiming at the realization of complex distributed dialog systems. In this article, we will report on the experience gained in the construction of these integrated large-scale research prototypes. The results obtained so far will be presented and the underlying principles of our approach will be described and discussed. An important practical result of our long-term work is MULTIPLATFORM, the **Mu**ltiple **L**anguage / **T**arget **I**ntegration **Plat**form **for M**odules (Herzog *et al.* 2003). The so-called MULTIPLATFORM testbed provides a powerful and complete integration platform, which is also freely available for external use.

First, the underlying principles of our proposed framework for large-scale dialog system integration will be presented and discussed. Section 3 then provides a detailed description of our realization of the approach within the MULTIPLATFORM testbed. In Section 4, we focus on an elaborate example for an XML-based specification of the data interfaces within a multimodal dialog system. Section 5 provides an overview of the various dialog systems that have been realized using our integration platform. After a discussion of related approaches in Section 6, the main results are summarized in the concluding section.

2 Foundations for an Integration Platform

Our notion of an integration platform comprises the software infrastructure that is needed to integrate heterogeneous dialog components into a self-contained system. The following sections highlight basic aspects and design principles that govern the construction of advanced natural-language and multimodal dialog systems.

2.1 Architecture Framework

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. In our project work, for example, we encountered modules for Sun Solaris, GNU Linux, and Microsoft Windows written in Prolog and Lisp, as classical AI languages, as well as in common conventional programming languages like C, C++, and Java.

The framework we propose is based on a component architecture (Hopkins, 2000) and our approach assumes a modularization of the dialog system into distinct and independent software modules to allow maximum decoupling. These large-grained components—ranging from more basic modules that encapsulate access to specific hardware devices to

complex components which may include entire application-specific subsystems—constitute self-contained applications which are executed as separate processes, or even process groups. The principle behind this view is to consider software architecture on a higher level of abstraction as modularization is not concerned with decomposition on the level of component libraries in a specific programming language. Continuous evolution is one of the driving forces behind the development of novel dialog systems. The creation of a modularized system makes the integrated application easier to maintain. In a well-designed system, the changes will be localized, and such changes can be made with little or no effect on the remaining components. 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 term middleware (Emmerich 2000) denotes the specific software infrastructure that facilitates the interaction among distributed software modules, i.e., the software layer between the operating system—including the basic communication protocols—and the distributed components that interact via the network. As a component platform, our integration framework has to enable inter-process communication and it needs to provide means for configuring and deploying the individual parts of the complete dialog system.

The details of the component architecture of different dialog systems vary significantly and an agreed-upon standard architecture which defines a definite modularization into separate processing units simply does not exist. Hence the precise specification of the overall system architecture constitutes an integral part of the joint realization process. Approaches like RAGS (Mellish *et al.* 2004), the **R**eference **A**rchitecture for **G**eneration **S**ystems, and the standard reference model for intelligent multimedia presentation systems described in (Bordegoni *et al.* 1997) as well as the architecture of intelligent user interfaces depicted in (Maybury and Wahlster 1998) provide valuable guidelines for laying out such a system architecture.

For our own design activities, we found it helpful to use a well-defined naming scheme and distinguish the following categories of dialog system components when designing a concrete system architecture:

Device: Connector modules that encapsulate access to a hardware component like, for example, microphone and sound card for audio input or a camera system that observes the user in order to enable the interpretation of facial expressions.

Recognizer: Modality-specific components that process input data on the signal level. Examples include speech recognition, determination of prosodic information, or gesture recognition.

Analyzer: Modules that further process recognized user input or intermediary results on a semantic level. Such components include in particular modality-specific analyzers and media fusion.

Generator: Knowledge-based components which determine and control the reactions of the dialog system. This includes the planning of dialog contributions and application-centric activities as well as fission of multiple modalities and media-specific generators, e.g., for text and graphics.

Synthesizer: Media-specific realization components that transform generated structures into perceivable output. A typical example is a speech synthesis component.

Modeller: Active knowledge sources that provide explicit models of relevant aspects of the dialog system, such as discourse memory, lexicon, or a model of the underlying application functionality.

Service: Connector components that provide a well-defined link to some application-specific functionality, like access to a TV program database. Service modules depend on the specific application scenario and often encapsulate complete and complex application-specific subsystems.

This coarse classification into distinct base categories supports modularization as it helps to detect components that may need further decomposition.

In terms of these component categories, the basic data flow within a typical dialog system can be described as follows: Parallel input streams from input devices through recognition and analysis components are followed by multiple analysis stages that result in a processable interpretation of the user input. The dialog management generates effects within the application system and initiates the generation of suitable output. The notion of different service components leads to a better modularization of the application interface. On the output side, multiple processing stages can be distinguished which involve different generation phases, synthesis components, and output devices. The processing is further supported by additional modeller components, which actively maintain dynamic knowledge sources that need to be shared between several processing components.

2.2 Inter-Process Communication

Nowadays, a very broad spectrum of practical technologies exists to realize communication between distributed software modules. Techniques like remote procedure call and remote method invocation which follow the client-server paradigm have long been the predominant abstraction for distributed processing. In this programming model, each component has to declare and implement a specific API to make its encapsulated functionality transparently available for other system modules. Only recently, the need for scalability, flexibility, and decoupling in large-enterprise and Internet applications has resulted in a strong general trend toward asynchronous, message-based communication in middleware systems.

In accordance with the long-standing distinction being made in AI between procedural vs. declarative representations, we favor message-oriented middleware as it enables more declarative interfaces between the components of a dialog system. As illustrated by a hybrid technology like SOAP, the *simple object access protocol*, where remote calls of object methods are encoded in XML messages, the borderline between a procedural and a declarative approach is rather difficult to draw in general.

For message-based communication, two main schemes can be distinguished:

- Basic *point-to-point* messaging employs unicast routing and realizes the notion of a direct connection between message sender and a known receiver. This is the typical interaction style used within multi-agent systems (Weiss 2000).
- 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 focuses on the data to be exchanged and it decouples data producers and data consumers. The well-known concept of a blackboard architecture, which has been developed in the field of AI (Erman *et al.* 1980), follows similar ideas.

In the context of our framework, we use the term *data pool* to refer to named message queues. An essential requirement is the ability to link every data pool to an individual data type specification in order to define admissible message contents.

In our view, the use of a multi-agent system and agent communication languages like KQML (Finin *et al.* 1994) or FIPA ACL (Pitt and Mamdani 1999) is not a premier choice for the proposed integration platform since in general, large-scale dialog systems are a mixture of knowledge-based and conventional data-processing components. A further aspect relates to the proposed data pool architecture, which does not rely on unspecific point-to-point communication but on a clear modularization of data links. Compared with point-to-point messaging, the publish/subscribe approach can help to reduce the number and complexity of interfaces significantly (Klüter *et al.* 2000). The specification of the content format for each pool defines the common language that dialog system components use to interoperate. The careful design of information flow and accurate specification of content formats constitute essential elements of our approach.

In the different dialog systems we designed so far, typical architecture patterns can be identified since the data pool structure reflects our classification into different categories of dialog components. The pool names, together with the module names, define the backbone for the overall architecture of the dialog system.

2.3 XML-based Data Interfaces

Over the last few years, the so-called *extensible markup language* (XML) has become the premier choice for the flexible definition of application-specific data formats for information exchange. XML technology, which is based on standardized specifications, progresses rapidly and offers an enormous spectrum of useful techniques and tools.

XML-based languages define an external notation for the representation of structured data and simplify the interchange of complex data between separate applications. All such languages share the basic XML syntax, which defines whether an arbitrary XML structure is well-formed, and they are built upon fundamental concepts like *elements* and *attributes*. A specific markup language needs to define the structure of the data by imposing constraints on the valid use of selected elements and attributes. This means that the language serves to encode semantic aspects of the data into syntactic restrictions.

Various approaches have been developed for the formal specification of XML-based languages. The most prominent formalism is called *document type definition*. A DTD basically defines for each allowed element all allowed attributes and possibly the acceptable attribute values as well as the nesting and occurrences of each element. The DTD approach, however, is more and more superseded by XML Schema. Compared with the older DTD mechanism, a schema definition (XSD) offers two main advantages: The schema itself is also specified in XML notation and the formalism is far more expressive as it enables more

detailed restrictions on valid data structures. This includes in particular the description of element contents and not only the element structure. Moreover it allows for context-specific specifications of element content instead of having just one global definition for a single element name.

To give a practical example, consider a tag like `<timestamp>` where the element definition in a DTD is only able to restrict the allowable contents to an unspecific sequence of characters. In an XSD, however, the string surrounded by `<timestamp>` tags can be constrained to provide a valid date and time specification.

As a schema specification can provide a well-organized type structure it also helps to better document the details of the data format definition. A reader-friendly presentation of the communication interfaces is an important aid during system development. This is particularly useful if the language definition reaches a substantial size. Our own data-oriented interface specifications, which will be discussed in more detail in Section 4, consist of more than 1,000 different XML elements.

It should be noted that the design of an XML language for the external representation of complex data constitutes a non-trivial task. Our experience is that design decisions have to be made carefully. For example, it is better to minimize 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.

From the point of view of component development, XML offers various techniques for the processing of transferred content structures. SAX, the *simple API for XML*, provides a standard event-based API for XML parsers. The DOM API makes the data available as a generic tree structure—the *document object model*—in terms of elements and attributes. 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 dialog 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 XML markup. In this approach, the language specification in form of a DTD or an XML Schema is exploited to automatically generate a corresponding object model in a given programming language.

2.4 Software Integration for Dialog Systems

The difficulty of system integration is high and often underestimated. Typical constraints for the construction of advanced natural-language and multimodal dialog systems include in particular the following aspects:

- Strong need to reuse and extend existing software;
- Pressure to build prototypes rapidly;
- Significant number (10–100) of distinct components from different sources;
- Mixture of knowledge-based components with reasoning capabilities and conventional software modules;
- Heterogeneous software environment (various programming languages and different operating systems).

An integration platform provides the necessary software infrastructure to assemble heterogeneous components into a complete dialog system. It can be characterized as a software development kit consisting of the following main parts:

- A run-time platform for the execution of a distributed dialog system;
- Application programming interfaces for participation in the run-time environment and for inter-process communication;
- Tools and utilities to support the whole development process, including also installation and software distribution.

Taking also into account the design principles outlined in the previous sections, it becomes obvious that available off-the-shelf middleware is not enough to provide a flexible architecture framework and a powerful integration platform for large-scale dialog systems. These considerations motivated our investment into the development of a suitable system integration platform.

In addition to the software infrastructure, the practical organization of the project constitutes a key factor for the successful realization of an integrated multimodal dialog system. Stepwise improvement and implementation of the design of architecture details and interfaces necessitate an intensive discussion process that has to include all participants who are involved in the realization of system components in order to reach a common understanding of the intended system behavior. Independent integration experts that focus on the overall dialog system have proven to be helpful for the coordination of this kind of activities.

Large projects like VERBMOBIL and SMARTKOM (see Section 5) have shown the need to take software engineering considerations in language technology projects seriously. For these projects, 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 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 VERBMOBIL and SMARTKOM projects is the consistent integration of a very large number of components created by diverse groups of researchers from disparate disciplines.

3 The MULTIPLATFORM Testbed

The MULTIPLATFORM testbed, or testbed for short, originates from our practical work in various research projects. The testbed, which is built on top of open source software

(Wu and Lin 2001), provides a fully-fledged integration platform for dialog systems and has been improved continually for many years now (Bub and Schwinn 1999; Klüter *et al.* 2000; Herzog *et al.* 2003).

3.1 Middleware Solution

Our specific realization of an integration platform follows the architecture framework and the design principles described in the previous section. So instead of using programming interfaces, the interaction between distributed components within the testbed framework is based on the exchange of structured data through messages.

MULTIPLATFORM includes a message-oriented middleware implementation that is based on PVM, which stands for *parallel virtual machine* (Geist *et al.* 1994). In order to provide publish/subscribe messaging with data pools on top of PVM, we had to add a further software layer called PCA (*pool communication architecture*).

The messaging system is able to transfer arbitrary data contents and provides excellent performance characteristics. To give a practical example, it is possible to perform a telephone conversation within a multimodal dialog system. 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. A typical multimodal user interaction of about 10 minutes duration can easily result in a message log that contains far more than 100 Megabytes of data.

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

Our middleware solution does not exclude to connect additional components during system execution. So far, however, the testbed does not offer specific support for dynamic system re-configuration at runtime. In our experience, it is acceptable and even beneficial to assume a stable, i.e., a static but configurable, architecture of the user interface components within a specific system instantiation. It is obvious that ad hoc activation and invocation of services constitutes an important issue in many application scenarios, in particular Internet-based applications. We propose to hide such dynamic aspects within the application-specific parts of the complete system so that they do not affect the basic configuration of the dialog system itself.

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 dialog system.

3.2 Testbed Modules and Offline Tools

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

The *testbed manager* component, or TBM for short, is responsible for system initialization and activates all distributed components pertaining to a given dialog 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. The state-transition diagram shown in Figure 1 forms the basis for this protocol, which is also directly supported by the API functions of the module manager interface.

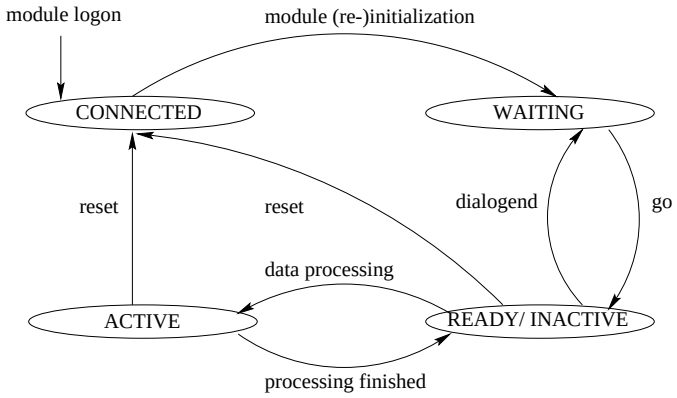


Fig. 1. Component states represented as state-transition network.

The freely configurable *testbed GUI* constitutes a separate component which provides a graphical user interface for the administration of a running system. In Figure 2 the specific testbed GUI of the SMARTKOM system (cf. Section 5.2) is shown as an example. The GUI basically provides 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.

A further logging component is being employed to save a complete protocol of all exchanged messages for later inspection. 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 3 presents an example of a word hypothesis graph. Such a graphical visualization of speech recognition results during system execution aids the system integrator while testing the dialog system.

The flexible data viewer can also be used in offline mode. Figure 4 shows a comprehensi-

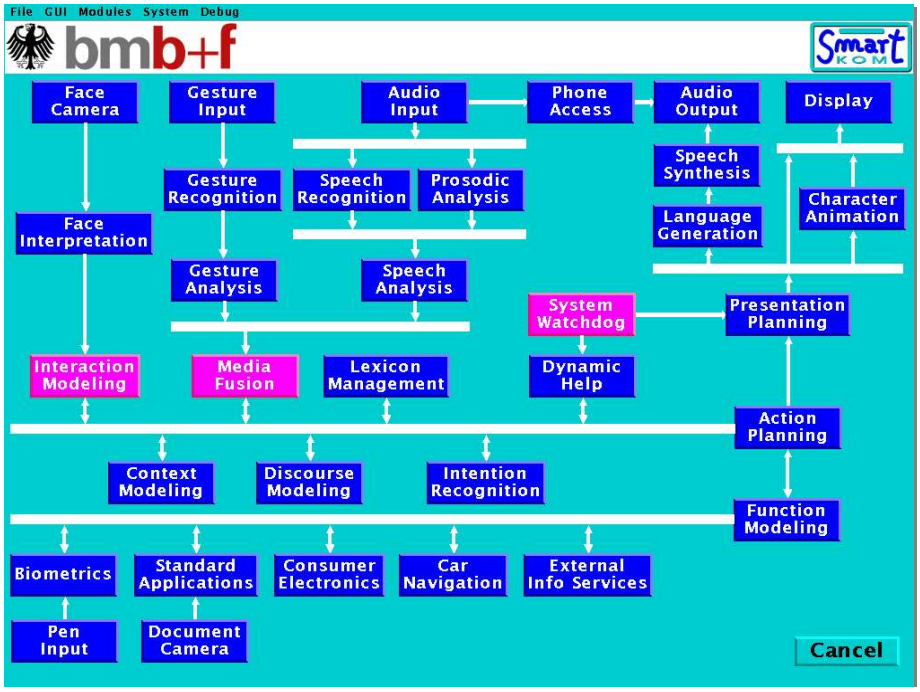


Fig. 2. Testbed administration GUI. Currently active components are highlighted using a different background color.

ble dialog protocol that documents an interaction with the SMARTKOM system. It provides a compact summary of important processing results and is generated fully automatically from the message log. Each item is prepended with the corresponding message number in order to enable more detailed inspection of relevant data.

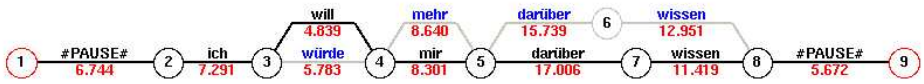
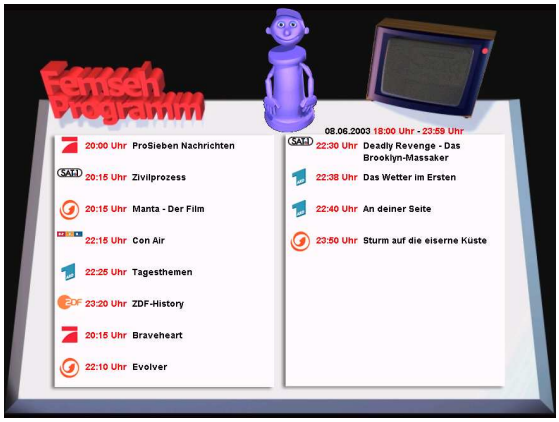
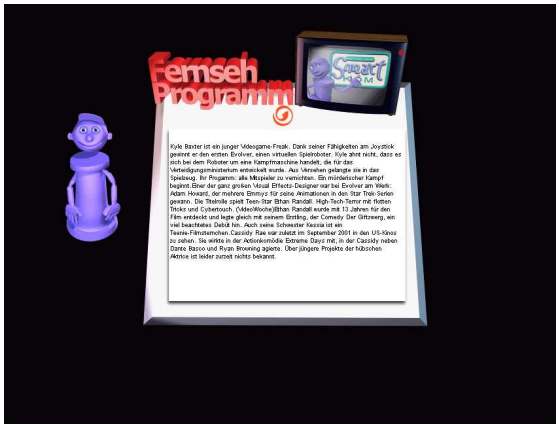


Fig. 3. Word hypothesis graph for “I would like to know more about this”.

Further offline tools include a standardized build and installation procedure for components and utilities for the preparation of software distributions and incremental updates during system integration. Additional project-specific APIs and specifically adapted utilities are being developed and made avaliable as needed.

...
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 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. 4. Excerpt from an automatically generated dialog protocol. The entries consist of the message number followed by the summarized content of selected data pools.

4 Multimodal Markup Language

Although MULTIPLATFORM is open to transfer all kinds of data contents between dialog system components, we prefer to operationalize interface specifications in the form of XML languages.

In the context of the SMARTKOM project (see Section 5.2) we have developed M3L (**M**ulti**m**odal **M**arkup **L**anguage) as a complete XML language that covers all data interfaces within this complex multimodal dialog system. Instead of using several quite different XML languages for the various data pools, we aimed at an integrated and coherent language specification, which includes all sub-structures that may occur on the different pools. In order to make the specification process manageable and to provide a thematic organization, the M3L language definition has been decomposed into about 40 schema specifications.

M3L is designed for the representation and exchange of complex multimodal content. It provides information about segmentation, synchronization, and the confidence in processing results. For example, gesture and word hypothesis graphs, hypotheses about facial expressions, media fusion results, and presentation goals are all represented in M3L. The XML schemas can be viewed as typed feature structures. For each data pool, they allow for automatic data and type checking during information exchange.

Figure 5 shows an excerpt from a typical M3L expression. It contains a sequence of interpretation hypotheses with the corresponding scores from speech and gesture recognition. A potential reference object for the deictic pointing gesture—the movie title *Evolver*—has been found in the visual context. The basic data flow from user input to system output continuously adds further processing results so that the representational structure will be refined step-by-step.

Intentionally, M3L has not been devised as a generic knowledge representation language, which would require an inference engine in every single component so that the exchanged structures can be interpreted adequately. Instead, very specific element structures are used to convey meaning on the syntactic level. Obviously, not all relevant semantic aspects can be covered on the syntax level using a formalism like DTD or XSD. This means, that it is impossible to exclude all kinds of meaningless data from the language definition and the design of an interface specification will always be a compromise.

Conceptual taxonomies provide the foundation for the representation of domain knowledge as it is required within a dialog system to enable a natural conversation in the given application scenario. SMARTKOM follows this approach with a domain specific ontology (Gurevych *et al.* 2003). In order to exchange instantiated knowledge structures between different system components they need to be encoded in M3L. Instead of relying on a manual reproduction of the underlying terminological knowledge within the M3L definition we decided to automate that task. Our tool OIL2XSD (Gurevych *et al.* 2003) transforms an ontology written in OIL (Fensel *et al.* 2001) into an M3L compatible XML Schema definition. The resulting schema specification captures the hierarchical structure and a significant part of the semantics of the ontology. The main advantage of this approach is that the structural knowledge available on the semantic level is consistently mapped to the communication interfaces and M3L can easily be updated as the ontology evolves.

For example in Figure 5, the representation of the event structure inside the intention

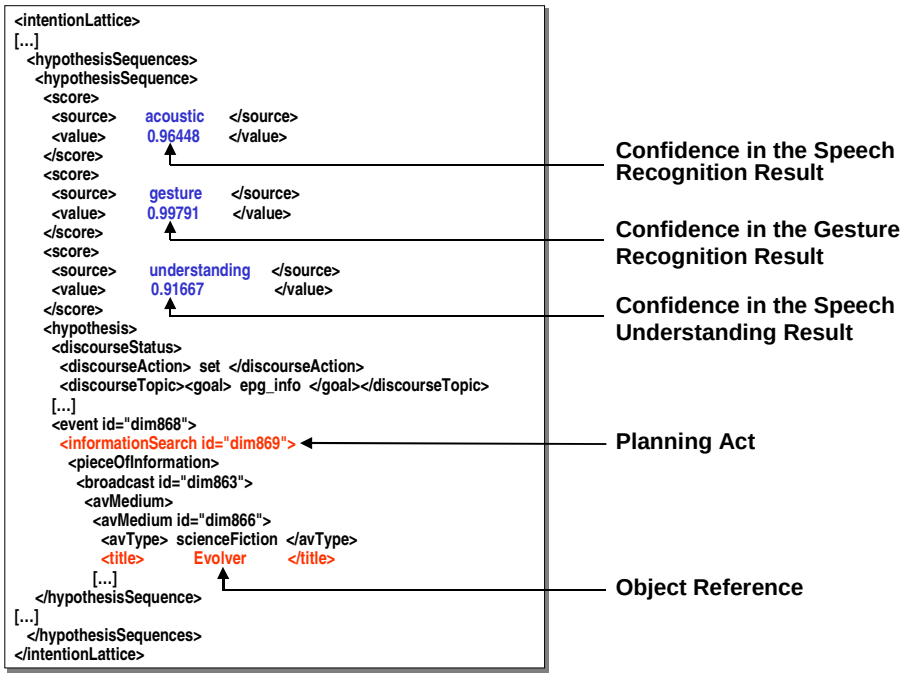


Fig. 5. Partial M3L structure. The shown intention lattice represents the interpretation result for a multimodal user input that can be stated as: "I would like to know more about this [↗]." The symbol [↗] represents a deictic pointing gesture to some object on the screen.

lattice originates from the ontology. Figure 6 presents the corresponding M3L element definition from the XML schema.

In addition to the language specification itself, a specific M3L API has been developed, which offers a light-weight programming interface to simplify the processing of such XML structures within the implementation of a component. Customized testbed utilities like tailored XSLT stylesheets for the generic data viewer as well as several other tools are provided for easier evaluation of M3L-based processing results.

5 Dialog System Applications

Our framework and the MULTIPLATFORM testbed have been employed to realize various natural language and multimodal dialog systems, ranging from medium-size to large-scale projects. In addition to the research prototypes mentioned here, MULTIPLATFORM has also been used as an integration platform for inhouse projects of industrial partners and for our own commercial projects.

More than 120 modules have already been used within the MULTIPLATFORM testbed. Given the fact that our framework does not prescribe the details of the data interfaces of specific dialog system components, these modules can not always be reused directly within

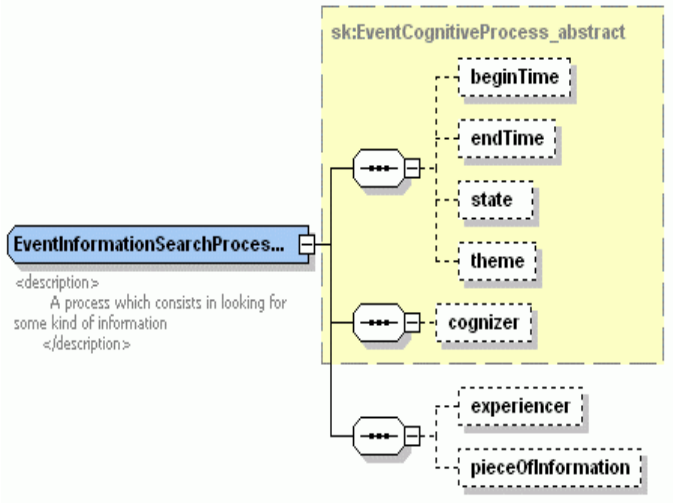


Fig. 6. Visualization of an element definition in an XML Schema. The shown part corresponds to an M3L event type that has been derived from the underlying ontology.

a different system context. In our experience, the module developers often have to make slight adaptations to cope with the architectural variations of new projects and their specific system designs.

5.1 Verbmobil

The first incarnation of MULTIPLATFORM arose from the VERBMOBIL project (Wahlster 2000) where the initial system architecture, which relied on a multi-agent approach with point-to-point communication, did not prove to be scalable (Klüter *et al.* 2000).

VERBMOBIL is a speaker-independent and bidirectional speech-to-speech translation system that aims to provide users in mobile situations with simultaneous dialog interpretation services for restricted topics. The system handles dialogs in three business-oriented domains—including appointment scheduling, travel planning, and remote PC maintenance—and provides context-sensitive translations between three languages (German, English, Japanese).

VERBMOBIL follows a hybrid approach that incorporates both deep and shallow processing schemes. A peculiarity of the architecture is its multi-engine approach. Five concurrent translations engines, based on statistical translation, case-based translation, substring-based translation, dialog-act based translation, and semantic transfer, compete to provide complete or partial translation results. The final choice of the translation result is done by a statistical selection module on the basis of the confidence measures provided by the translation paths.

In addition to a stationary prototype for face-to-face dialogs, another instance has been realized to offer translation services via telephone (Kirchmann *et al.* 2000).

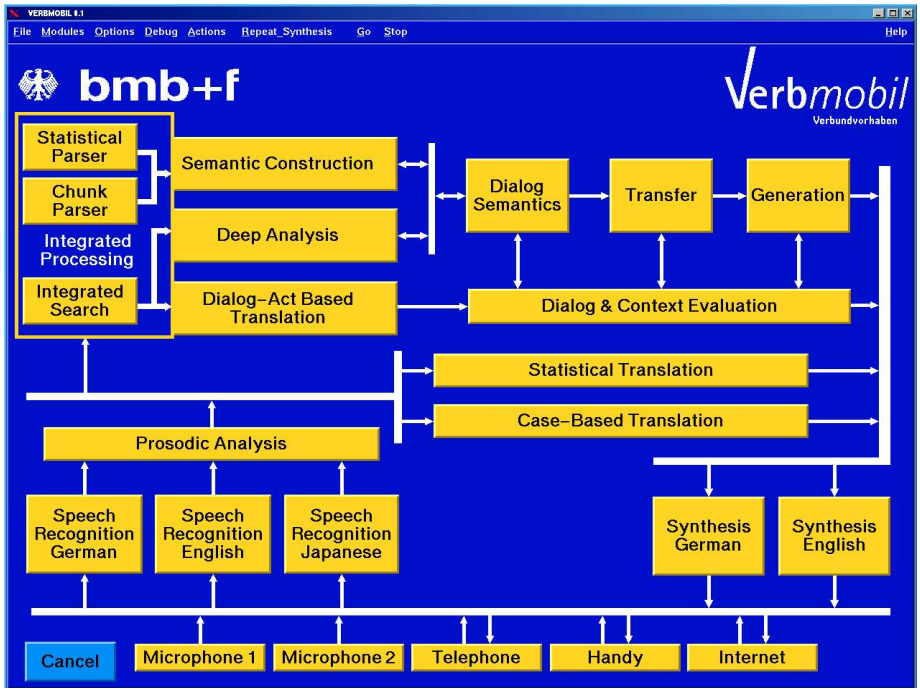


Fig. 7. VERBMOBIL's control GUI.

The final VERBMOBIL demonstrator consists of about 70 distributed software components that work together to recognize spoken input, analyze and translate it, and finally generate and utter the translation (cf. Figure 7). These modules are embedded into an earlier version of the MULTIPLATFORM testbed using about 200 data pools—replacing several thousand point-to-point connections—to interconnect the components.

5.2 SmartKom

In the context of the SMARTKOM project (Wahlster 2003; Wahlster *et al.* 2001) the testbed has been further enhanced. The decisive improvement of the current MULTIPLATFORM testbed is, besides a more robust implementation, a generalized architecture framework for multimodal dialog systems and the use of XML-based data interfaces as exemplified by the Multimodal Markup Language M3L.

SMARTKOM is a mixed-initiative dialog system that provides full symmetric multimodality by combining speech, gesture, and facial expressions for both, user input and system output (Wahlster 2003; Wahlster *et al.* 2001). The system aims to provide an anthropomorphic and affective user interface through its personification of an embodied conversational agent. The interaction metaphor is based on the so-called *situated, delegation-oriented dialog paradigm*. The basic idea is, that the user delegates a task to a virtual

communication assistant which is visualized as a life-like character. The interface agent recognizes the user's intentions and goals, asks the user for feedback if necessary, accesses the various services on behalf of the user, and presents the results in an adequate manner.

The current version of the MULTIPLATFORM testbed, including M3L, is used as the integration platform for SMARTKOM. The overall system architecture includes about 40 different components which can be distributed over several computers. The modules communicate using about 100 different data pools. As it is depicted in Figure 8, SMARTKOM realizes a flexible and adaptive shell for multimodal dialogs and addresses three different application scenarios:

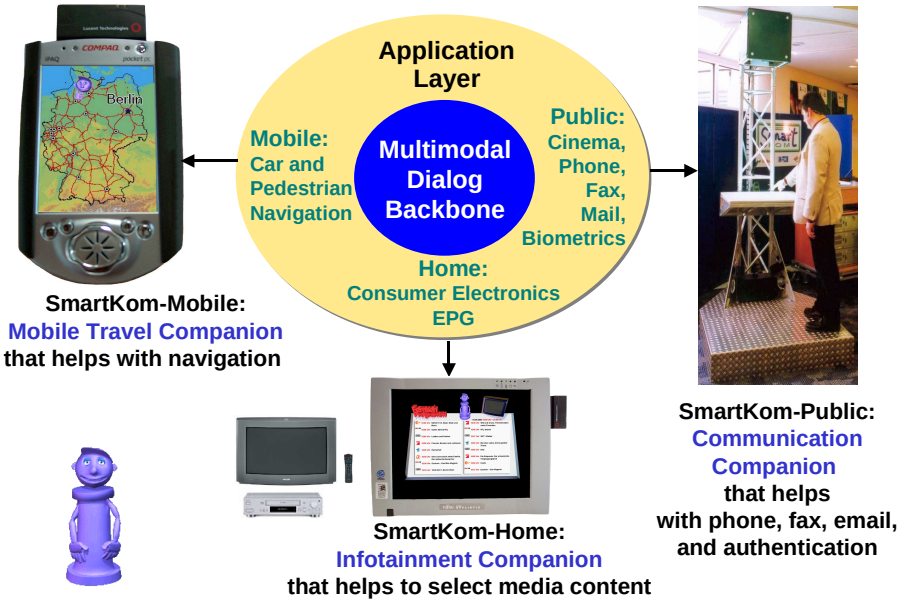


Fig. 8. SMARTKOM kernel and application scenarios. Smartakus, the SMARTKOM life-like character is shown in the lower left corner.

SMARTKOM PUBLIC realizes an advanced multimodal information and communication kiosk for airports, train stations, or other public places. It supports users seeking for information concerning movie programs, offers reservation facilities, and provides personalized communication services using telephone, fax, or electronic mail. Moreover, it helps users in the biometric authentication tasks (hand contour recognition, signature or voice verification), e.g., in a security-critical application. Speech input is captured with a directional microphone; facial expressions are captured with a DV camera and gestures are tracked with an infrared camera. An additional camera is used to capture images of documents or 3D objects the user wants to include in multimedia messages composed with the help

of SMARTKOM. A video projector is used for the projection of graphical output onto a horizontal surface.

SMARTKOM HOME serves as a multimodal infotainment companion that helps to select media content and to operate various appliances. Using a portable webpad, the user is able to utilize the system as an electronic program guide or to easily control consumer electronics devices like a TV set or a VCR. Similar to the kiosk application, the user may also use communication services at home. In the context of SMARTKOM HOME two different interaction modes are supported and the user is able to easily switch between them. In *lean-forward* mode coordinated speech and gesture input can be used for multimodal interaction with the system: the user interacts with the touchpad, paying attention to the generated presentation, listening to the spoken output and reacting by pen gestures and speech. *Lean-backward* mode instead is constrained to verbal communication. This mode is assumed when the user can not or is not willing to pay attention to the visual context and interacts solely by means of speech input and output.

SMARTKOM MOBILE realizes a mobile travel companion for navigation and location-based services. It uses a PDA as a front end, which can be added to a car navigation system or is carried by a pedestrian. This application scenario comprises services like integrated trip planning and incremental route guidance through a city via GPS and GSM, GPRS, or UMTS connectivity. In the mobile scenario speech input can be combined with pen-based pointing and a simplified version of the Smartakus interface agent combines speech output, gestures and facial expressions.

5.3 Comic

COMIC, which stands for **Co**nversational **M**ultimodal **I**nteraction with **C**omputers, is a recent research project that focuses on computer-based mechanisms of interaction in cooperative work.One specific sample application for COMIC is a design tool for bathrooms with an enhanced multimodal interface (Catizone *et al.* 2003). The main goal of the experimental work is to show that advanced multimodal interaction based on generic cognitive models for man-machine interaction can make such a tool usable for non-experts as well.

The realization of the integrated COMIC demonstrator is based on the latest version of the MULTIPLATFORM testbed. Figure 9 shows the control interface of the multimodal dialog system, which currently includes 12 modules from 7 research groups and uses about 30 data pools.

On the input side, speech, gestures, drawing and handwriting can be employed in combination by the user. On the output side, a computer-generated, realisitic face with synthesized expressions, head and eye movements is combined with task-related graphical, textual and spoken information. Dialog languages are English or German. In addition to multiple input and output channels and the corresponding recognition and synthesis modules, there are dialog-backbone modules for analysis, including multimodal fusion, a dialog and action managment module, a fission module to distribute the output information over available and appropriate modalities and a natural language generation module.

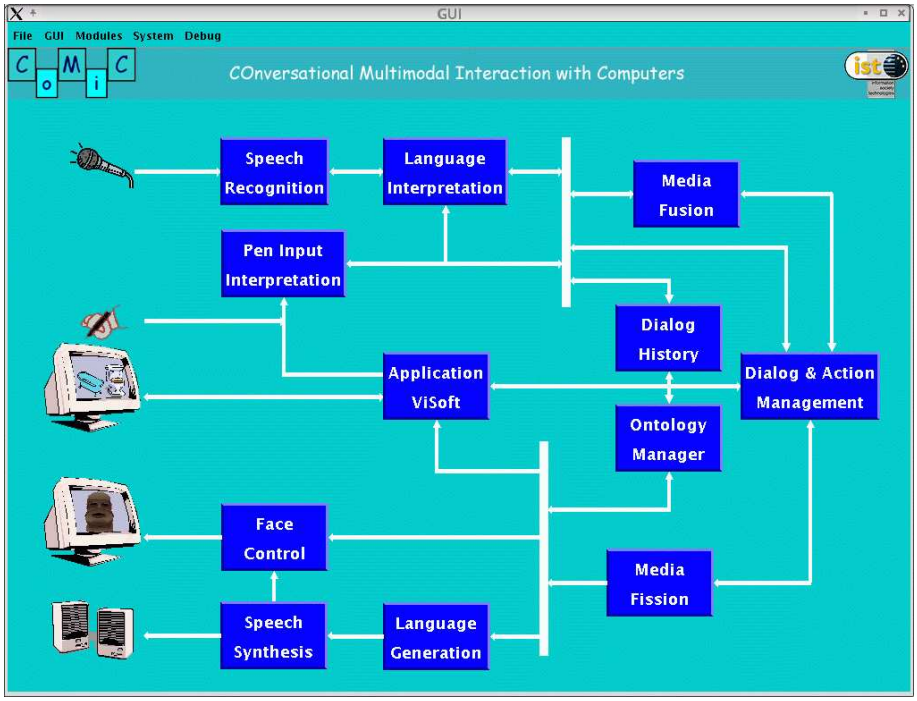


Fig. 9. Adapted testbed GUI for the COMIC system.

5.4 AddVoice

A further research prototype that employs MULTIPLATFORM as integration testbed is the Philips ADDVOICE system. ADDVOICE features a set of applications similar to those described in SMARTKOM HOME (Portele *et al.* 2003). It is based upon the SPICE (**S**peech **I**nterfaces for **C**onsumer **E**lectronics) prototype system, representing a multimodal conversational user interface to an electronic program guide (Kellner and Portele 2002). Its purpose is to simplify the access to the devices in a home environment (e.g., controlling a TV set or programming a VCR) and to support the user in the navigation in a large TV program database.

The interaction with the system is based on natural language input in combination with pointing facilities through a touch-screen on a hand-held graphical display. On the output side, the system uses spoken feedback based on template-based language generation as well as graphical illustrations. A central role within the specific architecture, which is reflected in the screen shot shown in Figure 10, is played by the dialog management component. The dialog manager is able to deal with disambiguation of imprecise user input, clarification of user intentions, and automatic relaxation if no match is found in the database.

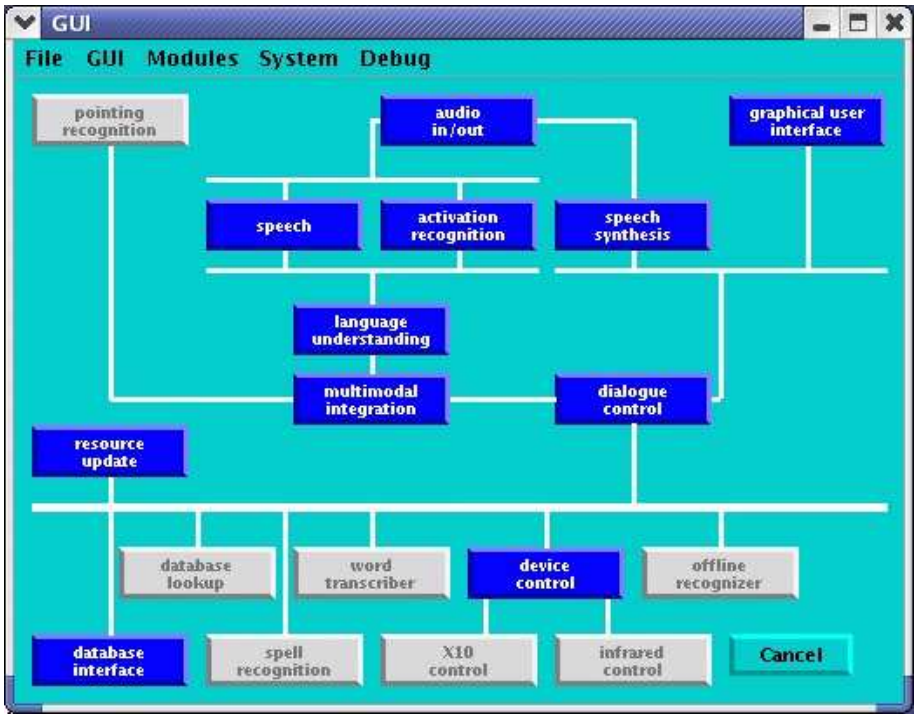


Fig. 10. Control interface of the ADDVOICE system.

6 Related Approaches

GCSI (Seneff *et al.* 1999) and OAA (Cheyer and Martin 2001; Martin *et al.* 1999) are two systems that are very similar in scope to the MULTIPLATFORM testbed. Both approaches provide complete integration platforms and constitute viable alternatives to our software solution.

GCSI, the **G**alaxy **C**ommunicator **s**oftware **i**nfrastructure, is an open source architecture for the realization of natural language dialog systems. It can be summarized as a distributed, message-based, client-server architecture, which has been developed for constructing spoken dialog systems. The key component in this framework is a central hub, which mediates the interaction among various servers that realize different dialog system components. The central hub does not only handle the communication among the server modules but is also responsible to maintain the flow of control that determines the processing within the integrated dialog system. To achieve this, the hub is able to interpret scripts encoded in a special purpose, run-time executable programming language. Since it would not be efficient to transfer all communication between servers via the hub, GCSI also supports peer-to-peer connections. The hub mediates the establishment of this kind of connection, but all data which flows through the connection goes directly from server to server.

OAA, the **O**pen **A**gent **A**rchitecture, is a framework for integrating a community of heterogeneous software agents in a distributed environment. All communication and cooperation between the different components is achieved via messages expressed in ICL, a logic-based declarative language capable of representing natural language expressions. Similar to the GCSI architecture, a sort of centralized processing unit is required to control the behavior of the integrated system. So-called facilitator agents reason about the agent interactions necessary for handling a given complex ICL expression, i.e., the facilitator coordinates the activities of agents for the purpose of achieving higher-level, complex problem-solving objectives. Sample applications built with the OAA framework also incorporated techniques to use multiple input modalities. The user can point, speak, draw, handwrite, or even use a standard graphical user interface in order to communicate with a collection of agents.

A fundamental difference between our architecture framework and both, GCSI as well as OAA, is a local instead of a global organization of the processing within the overall system. Within the MULTIPLATFORM testbed there exists no centralized controller component which could become a potential bottleneck for more complex dialog systems. GCSI includes the option to employ low-level communication links in order to transfer data directly from one component to another. This feature, however, leads to an intransparent system that is more difficult to integrate and test since the flow of information can not be fully monitored.

GATE, the **G**eneral **A**rchitecture for **T**ext **E**ngineering (Bontcheva *et al.* 2004; Cunningham *et al.* 2002), focuses more on text than on spoken language. It aims to provide a general infrastructure which allows users to build and customize natural language processing components. All communication between the components of a system modeled by GATE goes through the so-called GDM, GATE Document Manager, which is a central database repository that stores all the information the system components generate about the documents they process and exchange. The GDM encapsulates the components and provides them with a uniform API for manipulating the data they produce and consume.

The RAGS approach (Cahill *et al.* 2000; Mellish *et al.* 2004), which stands for **R**eference **A**rchitecture for **G**eneration **S**ystems, concentrates on natural language generation systems and aims to produce an architectural specification and model for the development of new applications in this area. RAGS is based on the well-known three-stage pipeline model for natural language generation which distinguishes between content determination, sentence planning, and linguistic realization. The main component of the RAGS architecture is a data model, in the form of a set of declarative linguistic representations which cover the various levels of representation that have to be taken into account within the generation process. XML-based notations for the data model can be used in order to exchange RAGS representations between distributed components. The reference architecture is open with respect to the technical interconnection of the different components of a generation system. One specifically supported solution is the use of a single centralized data repository.

Compared with our coarse-grained component framework, RAGS provides a fine-grained architecture model for natural language generation. A language generator within a system based on MULTIPLATFORM is usually realized as a single, integrated component, like for example the generation component in SMARTKOM. The internal architecture of this

specific language generator nethertheless follows the principles of RAGS as a reference architecture.

7 Conclusion

MULTIPLATFORM provides a practical framework for large-scale software integration that results from the realization of various natural language and multimodal dialog systems. The MULTIPLATFORM testbed is based on an open component architecture which employs message-passing to interconnect distributed software modules. We propose to operationalize interface specifications in the form of an XML language as a viable approach to assemble knowledge-based as well as conventional components into an integrated dialog system.

Our technical realization follows basic design principles which seem to be valuable for any integration platform. Asynchronous message-passing is more flexible than component-specific remote APIs—using for example RPC, CORBA, or even SOAP—and better fits parallel processing in a distributed system. Blackboard-like publish/subscribe messaging is a generalization of point-to-point communication which supports decoupling of components and allows better modularization of data interfaces. The proposed framework does not rely on centralized control. Individual components take care if coordination is needed and required synchronization information is modelled explicitly within data interfaces. There is no hidden interaction between components. All data is exchanged transparently via the provided communication system.

An open source version of the MULTIPLATFORM software has been made available at:

`http://multiplatform.sourceforge.net/`

References

Bontcheva, Kalina, Tablan, Valentin, Maynard, Diana, and Cunningham, Hamish (2004) Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, **10** (3/4), 349–373.

Bordegoni, Monica, Faconti, Giorgio, Feiner, Steven, Maybury, Mark T., Rist, Thomas, Ruggieri, Salvatore, Trahanias, Panos, and Wilson, Michael (1997) A Standard Reference Model for Intelligent Multimedia Presentation Systems. *Computer Standards & Interfaces*, **18**(6–7), 477–496.

Bub, Thomas and Schwinn, Johannes (1999) The Verbmobil Prototype System—A Software Engineering Perspective. *Natural Language Engineering*, **5**(1), 95–112.

Cahill, Lynne, Doran, Christy, Evans, Roger, Kibble, Rodger, Mellish, Chris, Paiva, Daniel, Reape, Mike, Scott, Donia, and Tipper, Neil (2000) Enabling Resource Sharing in Language Generation: An Abstract Reference Architecture. *In: Proc. of the 2nd Int. Conf. on Language Resources and Evaluation*.

Catizone, Roberta, Setzer, Andrea, and Wilks, Yorick (2003) Multimodal Dialogue Management in the COMIC Project. *In: Proc. of the EACL-03 Workshop on 'Dialogue Systems: Interaction, Adaptation and Styles of Management'*. European Chapter of the Association for Computational Linguistics, Budapest, Hungary.

Cheyer, Adam J., and Martin, David L. (2001) The Open Agent Architecture. *Autonomous Agents and Multi-Agent Systems*, **4**(1–2), 143–148.

Cunningham, Hamish and Patrick, Jon (eds.) 2003, *HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS)*. Edmonton, Canada: Association for Computational Linguistics.

Cunningham, Hamish, Maynard, Diana, Bontcheva, Kalina, and Tablan, Valentin (2002) GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. *In: Proc. of the 40th Anniversary Meeting of the ACL*. Philadelphia, PA: Association for Computational Linguistics.

Emmerich, Wolfgang (2000) Software Engineering and Middleware: A Roadmap. *In: Proc. of the Conf. on the Future of Software Engineering*, Limerick, Ireland, pp. 117–129. ACM Press.

Erman, Lee D., Hayes-Roth, Frederick, Lesser, Victor R., and Reddy, D. Raj (1980) The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Surveys*, **12**(2), 213–253.

Fensel, Dieter, van Harmelen, Frank, Horrocks, Ian, McGuinness, Deborah L., and Patel-Schneider, Peter F. (2001) OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, **16**(2), 38–45.

Finin, Tim, Fritzson, Richard, McKay, Don, and McEntire, Robin (1994) KQML as an Agent Communication Language. *In: Proc. of the 3rd Int. Conf. on Information and Knowledge Management*, Gaithersburg, MD, pp. 456–463. ACM Press.

Geist, Al, Beguelin, Adam, Dongorra, Jack, Jiang, Weicheng, Manchek, Robert, and Sunderman, Vaidy (1994) *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press.

Gurevych, Iryna, Merten, Stefan, and Porzel, Robert (2003) Automatic Creation of Interface Specifi cations from Ontologies. *In:* (Cunningham and Patrick 2003), pp. 59–66.

Herzog, Gerd, Kirchmann, Heinz, Merten, Stefan, Ndiaye, Alassane, Poller, Peter, and Becker, Tilman (2003) MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems. *In:* (Cunningham and Patrick 2003), pp. 75–82.

Hopkins, Jon (2000) Component Primer. *Communications of the ACM*, **43**(10), 27–30.

Kellner, Andreas and Portele, Thomas (2002) SPICE—A Multimodal Conversational User Interface to an Electronic Program Guide. *In: Proc. of the ISCA Tutorial and Research Workshop on Multi-Modal Dialogue in Mobile Environments*, Kloster Irsee.

Kirchmann, Heinz, Ndiaye, Alassane, and Klüter, Andreas (2000) From a Stationary Prototype to Telephone Translation Services. *In:* (Wahlster 2000), pp. 659–669.

Klüter, Andreas, Ndiaye, Alassane, and Kirchmann, Heinz (2000) Verbmobil From a Software Engineering Point of View: System Design and Software Integration. *In:* (Wahlster 2000), pp. 635–658.

Leidner, Jochen (2003) Current Issues in Software Engineering for Natural Language Processing. *In:* (Cunningham and Patrick 2003), pp. 45–50.

Martin, David L., Cheyer, Adam J., and Moran, Douglas B. (1999) The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, **13**(1–2), 91–128.

Maybury, Mark T. and Wahlster, Wolfgang (eds.) 1998, Intelligent User Interfaces. San Francisco: Morgan Kaufmann.

Mellish, Chris, Reape, Mike, Cahill, Lynne, Evans, Roger, Paiva, Daniel, and Scott, Donia (2004) A Reference Architecture for Generation Systems. *Natural Language Engineering*, **10** (3/4), 227–260.

Pitt, Jeremy and Mamdani, Abe (1999) Some Remarks on the Semantics of FIPA's Agent Communication Language. *Autonomous Agents and Multi-Agent Systems*, **2**(4), 333–356.

Portele, Thomas, Gorony, Silke, Emele, Martin, Kellner, Andreas, Torge, Sunna, and te Vrugt, Jürgen (2003) SmartKom-Home—An Advanced Multi-Modal Interface to Home Entertainment. *In: Proc. of Eurospeech'03*, Geneva, Switzerland.

Seneff, Stephanie, Lau, Raymond, and Polifroni, Joseph (1999) Organization, Communication, and Control in the Galaxy-II Conversational System. *In: Proc. of Eurospeech'99*, Budapest, Hungary, pp. 1271–1274.

Wahlster, Wolfgang (ed.) 2000. *Verbmobil: Foundations of Speech-to-Speech Translation*. Berlin: Springer.

Wahlster, Wolfgang (2003) SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell. *In:* Krahl, R., and Günther, D. (eds), *Proc. of the Human Computer Interaction Status Conference 2003*, pp. 47–62. Berlin, Germany: DLR.

Wahlster, Wolfgang, Reithinger, Norbert, and Blocher, Anselm (2001) SmartKom: Multimodal Communication with a Life-Like Character. *In: Proc. of Eurospeech'01*, Aalborg, Denmark, pp. 1547–1550.

Weiss, Gerhard (ed.) 2000, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press.

Wu, Ming-Wei and Lin, Ying-Dar (2001) Open Source Software Development: An Overview. *IEEE Computer*, **34**(6), 33–38.