Centrally Managed System Infrastructure and Integration in COMIC

Peter Poller

German Research Center for Artificial Intelligence (DFKI GmbH),
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany

Abstract. The following article reports on centrally managed integration of distributedly developed software components into the multi-modal dialog system COMIC. First, the technical issue of selecting and providing an appropriate inter-modular communication infrastructure is briefly illustrated. Second, the organization of the integration of middleware compliant software components into a fully functional multi-modal dialog system is recapitulated and described in detail. Finally, the experiences gained and the results of these procedures are compiled into a list of the most important aspects for further disposition in other contexts.

1 Introduction

In the last 7 years, DFKI played a central role in three large-scale projects focusing on different aspects of multi-modal dialogue systems: SMARTKOM[1] (09/1999 - 09/2003), COMIC[2] (03/2002 - 02/2005), and TALK[3] (01/2004 - 12/2006). In this paper, we report on conclusions and recommendations drawn from experiences and observations on centrally managed infrastructure and the four system integration cycles in the COMIC project. First, we give an introduction of the COMIC demonstrator. Then, we concentrate on the essentials needed to implement the system, i.e, communication infrastructure and system integration. Finally, we present the results, recommendations, and conclusions resulting from of such a complex implementation and integration task.

2 The COMIC System

COMIC was a three years R&D project concentrating on research and implementation of speech centric multi-modal conversational systems within a consortium of six different research institutes from three countries and one small SME. The final COMIC demonstrator is a multi-modal dialog system that helps the users to design their new bathroom within a natural dialog around a bathroom design application. The system comprises 15 independent software components (also

[1] SMARTKOM was funded by the German Federal Ministry for Education, Science, Research and Technology (BMBF) under grant 01 IL 905 A5.
[2] COMIC was funded by the European Union in FP5 under grant IST-2001-32311.
[3] TALK is also funded by the European Union in FP6 under grant IST-2003-507802.

referred to as modules) that communicate over 34 different inter-modular communication channels (sc. data pools). Figure 1 shows a screenshot of the COMIC *control GUI* in which each module is reflected as a blue button while their interfaces are represented by the white lines in a simplified fashion. It is a multifunctional tool for (i) visualization and demonstration purposes, (ii) comfortable system configuration and start-up, and (iii) module control and resetting.

COMIC runs in a distributed fashion on two Linux PCs and one Windows PC with one output screen each. The interaction with the system (see figure 2) is a speech centered multi-modal dialogue in terms of a conversation with the face – an animated agent consisting of a face that performs lip-synchronous mouth movements and mimics running under Linux – complemented by simultaneous pen drawings and gestures on the graphical output presentations of the *ViSoft*[4] application on a tablet Windows screen. The *control GUI* runs on an auxiliary screen for supervising purposes only.

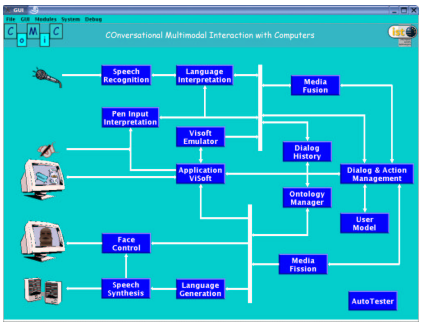

Fig. 1. COMIC *control GUI*

Fig. 2. *User interaction in* COMIC

3 Infrastructure

The functional system architecture of COMIC is based on a traditional modularization into distinct and independent software components for multi-modal dialog systems comprising, e.g, device control, recognition, analysis, generation, synthesis, and the *ViSoft* application. The first technical implementation task was the provision of an optimally suitable inter-modular communication infrastructure. The spectrum of candidates ranged from basic interfaces following a client-server paradigm to higher-level abstracting communication frameworks. Due to the modularized software division, we decided at a very early stage to use a suitable higher-level middleware architecture framework in order to benefit

[4] The *ViSoft* tool (http://www.visoft.de) comprises large databases of tiles and sanitary ware from manufacturers to choose from and generates dimensionally accurate, photo-realistic 3D views of respectively equipped bathrooms.

from its advantages: (i) efficient inter-process communication infrastructure on an abstract higher-level communication model (including a well-defined module API), and (ii) useful tools that support system development and handling, including utilities for debugging, configuring, integration, and testing purposes.

The remainder of the paper focuses on our specific experiences with MULTIPLATFORM in the COMIC project, but these are currently exploited for the Open Agent Architecture [5, 2] in the TALK project and should be transferable further to other projects with comparable implementation tasks as well.

3.1 Selection of a Middleware Architecture

The selection of a middleware infrastructure is a crucial step because it's impossible to revise it later on in the course of the project. In COMIC we took 4 month of preparation conducting the following 4 steps:

1. Extensive compilation of a detailed hardware and software overview of all existing and planned system components.
2. Appraisal of appropriate infrastructure platforms or frameworks.
3. Estimation of the effort to provide a project specific platform variant.
4. Comparison and decision.

The module specific features and requirements of each project partner and each software component were collected by means of a corresponding questionnaire. These questionnaires brought to light that the first COMIC demonstrator will be a distributed system running on the two operating systems Linux and Windows, initially comprising 9 software components programmed in three different programming languages. Realistically, the number of components cannot be considered fixed for the rest of the project. In COMIC it grew from 9 up to 15. Obviously, growing functionalities naturally require additional software components. Altogether, the mandatory infrastructure requirements were:

- Flexibility concerning the number and the contents of inter-modular communication channels as well as the number of modules at all.
- Module APIs for C, C++, and Java.
- Distributed processing with bi-directional communications on several machines under Linux and Windows.
- Flexible configurability, i.e., operability in arbitrary module combinations.
- Supporting tools and utilities.

To match these constraints, we shortlisted the Open Agent Architecture (OAA) [5, 2], the Galaxy Communicator [1], and MULTIPLATFORM [3, 4]. Apart from the differing communication models, our attention concentrated on the following practically relevant parameters and constraints which may be easily transferable to other projects in similar situations:

- Technical acceptability concerning performance, flexibility, stability, and configurability.

- Availability of tools and utilities for debugging, installation, starting up, monitoring, logging, log-data filtering, and testing.
- Declarativity as far as possible for, e.g., specification of communication channels and individual system configurations.
- Extent and Completeness of the documentation (module API, system handling, tools).
- Availability of the platform beyond the end of project.
- Experience at project partners.
- Minimal effort to provide a project specific platform (if necessary at all).
- Availability of source code for rapid bug fixing (optional).
- Disposability of support, e.g., from platform owner or via active mailing lists (optional).

Based on a comparison according to the criteria above, the project partners voted for MULTIPLATFORM as the best suitable framework. Nevertheless, the preparation of a project specific variant of MULTIPLATFORM took extra, but well invested efforts, e.g., as it provided project specific middleware tools like the *control GUI* (see figure 1 above) for visualization and control purposes. Furthermore, these extra efforts were more than equated by the benefits from DFKI's practical (positive and negative) experiences in SMARTKOM:

- Reuse of an easily adaptable module developer tutorial.
- Well founded module developer guidelines containing directives about
  - Module packaging
  - Directory organization (for the overall system and the modules)
  - Module versioning.
- Central version specifications for operating systems and programming languages.
- Data producers are strictly responsible for content and maintenance of their output data format (SMARTKOM: High effort for central data format supervision).
- Practical organization and management of system implementation and integration described in the following sections.

4 Integration

In COMIC we integrated four continously improved demonstrator systems. These integration cycles were conducted in line with the following general principles:

- Data and modules are exchanged by means of a central ftp-server.
- All modules must be centrally approved and released by the integration manager before distribution in the consortium.
- Every partner must have the whole system installed.
- Every partner must regularly update its system according to module releases.
- The integration manager supervises the (technical) correctness of the modules and manages their updating.

4.1 Management Tools for Monitoring

An important part of centrally managed system integration is the monitoring of the respective state of affairs. In COMIC, we distinguished right from the start between two aspects of the integration: (i) *technical*, and (ii) *functional*. *Technical* matters exclusively concern MULTIPLATFORM compliance aspects, e.g., the logon procedure or the purely technical capability to correctly handle incoming messages while *functional* matters are aspects of the contents, e.g., the correct I/O-behavior with respect to the communicated data, or the correct functionality of a module or the overall system. Consequently, we used different tools for monitoring technical and functional aspects during an integration.

For technical matters we developed a special table, the sc. *integration matrix* in which every module was assigned to a line. The overall technical state of a module was signaled by colored lines in the traffic light colors red, yellow, and green while the details of wrong or missing features were described in dedicated table columns. System releases were conducted by means of the announcement of an updated integration matrix within the whole consortium.

For functional matters, we used several bug tracking procedures. We started with a central management by the integration manager for module and system functionality aspects and appropriate bug assignments via E-mails. The intention to exceptionally manage all aspects of the first integration centrally was to guide once the whole consortium through a complete integration. Later on, we followed different concepts of distributing the functional responsibility within the whole consortium which are described in more detail in section 4.4.

Generally speaking, we tried to balance the often observed "trade-off" between (i) providing insufficient information which could slow down partner activities due to lacking information, and (ii) providing a constant stream of potentially bothering information which could have the same effect when important information is read over. Therefore, we decided for a mix, i.e., central monitoring of the technical integration and dynamically emerging functional monitoring.

4.2 Initial Integration

The very first system integration is singled out here as it naturally required more effort within the consortium than the following integration cycles. First of all, every partner must have successfully installed the middleware packages. Second, every partner must be capable of providing middleware-compliant software components consistent with the predefined module developer guidelines. Third, due to the lack of some of the components we must rely partly on manually edited example data to simulate inter-modular communication.

According to a previously approved time plan, the integration started with the delivery of the appropriate MULTIPLATFORM testbeds for the two operating systems. Every partner was obliged to properly install them. Then we interconnected a "dummy module integration" with the aim to make sure that every partner is capable to correctly implement and deliver a technically correct but functionally empty MULTIPLATFORM compliant component.

The real integration itself was organized as a stepwise delivery procedure on a weekly base for the individual modules that was distributed over 5 weeks. We organized the order of deliveries according to the processing within the system, i.e., we started at the input modules and ended at the output modules. The planned functionality of the overall system was clearly specified in advance by means of detailed example dialogs (including realistic interface data for the respective communication channels). The tasks of the integration manager were (i) the technical supervision of the modules and their interactions, (ii) the release of modules and system updates for the whole consortium, and (iii) manual testing of predefined example data. Finally, he supervised the overall system functionality by intensive testing and assignment of responsibilities for identified bugs.

4.3 System and Module Testing

As indicated above, appropriate testing procedures of various system aspects is an important task. Again, we distinguish between technical and functional testing. In addition to that, there is an orthogonal distinction between module-oriented and system-oriented testing.

Module-oriented and system-oriented technical testing was centrally conducted by the integration manager. The technical approval of a module consisted of a manual check of compliance with the predefined module developer guidelines.

Functional module-oriented testing was of course assigned to the module developers themselves. Naturally, it should be obvious that a module can only be delivered if it has the expected functionality and this functionality had explicitely been verified by the module developer before delivery.

Finally, functional system-oriented testing requires by far the most effort because local testing does not reveal all remaining bugs. As mentioned before, for the first integration this task was also assigned to the integration manager. In the following integration cycles it was distributed within the consortium in several ways as described in the following section.

4.4 Follow-Up Integration Cycles

In the following three integration cycles we sightly modified the integration procedures in order to improve them based on the experiences with the preceeding integration cycles and the growing expertise within the consortium.

For the functional specification of the demonstrators we continously followed the concept of extensively specified example dialogs. Technical integration issues were clearly distinguished from functional ones. In all integration cycles we followed the concept of announcing integration matrices to regularly reflect the latest technical state of the modules. Concerning time plans, the stepwise integration on a weekly base was retained from the initial integration.

For the second demonstrator, we followed the functional integration concept named "turn of the week" which we had successfully used in SmartKom. A

specific turn was taken from one of the example dialogs and proclaimed as the implementation goal for one week with the aim to achieve the complete processing of this turn starting from the user input until the corresponding system output within that week. In COMIC, the concept "turn of the week" turned out to be impractical. At the beginning of the second integration it was too ambitious to realize one system turn per week while at the end of that integration – in the course of increasing system capabilities – the realization of one system turn got a one day task.

At the same time, a new testing tool, the sc. *autoTester*, was implemented to semi-automatically perform functionally relevant testsuites based on the processing of previously stored syntactically correct example data of different kinds of test sets containing perfect as well as realistic data with respect to speech recognition quality. The *autoTester* captures the resulting data sets of selected other interfaces and puts them into an appropriate format. This way, every partner using the *autoTester* automatically worked with the same testsuites.

Consequently, for the third demonstrator, we intensified the distribution of the testing responsibility within the consortium by taking turns between partners on a weekly base. Thereby we achieved collaborative distributed testing, debugging and immediate bug fixing in front the running system. Bugs and errors were immediately published by E-mail within the consortium while the corresponding log-data for reproducing a bug were put on the central ftp server due to the huge amount of data (10 minutes of system interaction produce up to 100 MB of log-data). Every partner was obliged to provide a new bug report at takeover and at the end of the respective week. A side effect of this procedure was that it automatically forced every partner to regularly update its local system installation (at least when responsible for testing or when a bug was assigned) in order to be able to recapitulate the published bugs at its local system installation.

For the final integration we were even able to overlap the time plans for integration and system testing because the final demonstrator was an improved version of the third demonstrator. Again, we followed the concept of distributing the testing responsibility within the consortium. Our distribution strategy followed the approach to make that partner responsible for testing who had just delivered a module or who was about to deliver a module.

All but the first integration was finished by a common fine-tuning meeting with all module developers in the same room for one or two days which significantly accelerated progress once more.

5 Conclusion and Recommendations

All in all the integration efforts for the four demonstrator (numbers for the individual demonstrators in parentheses)

- Module deliveries: 434 (74, 160, 100, 100)
- Integration matrices: 115 (24, 50, 30, 11)

- System tests: 1700 (300, 600, 400, 600)
- E-mails: 4400 (600, 2000, 800, 1000)
- Conversations: 310 (80, 150, 80, 100)

These numbers show a few interesting trends. First, the effort for the initial integration was relatively moderate because we interconnected the "dummy module integration" whose efforts are not included above and the system functionality was very restricted compared to the following demonstrators. Beginning with the second integration there is obviously a trend to increasing integration efficiency according to the growing expertise within the consortium. Interestingly, this affects the number of module deliveries only while the increasing complexity of the demonstrators come along with equally growing needs for testing. So, in the end it turns out that the integration management effort was not for nothing, it pays off.

We conclude with a list of the most important aspects that should be carefully considered in any comparable situation.

- Well founded selection of the infrastructure middleware.
- Awareness of the special role of the initial integration.
- Careful reviewing of the planned monitoring tools and aspects.
- Well elaborated realistic integration time plans.
- Previously agreed-on releasing procedures.
- Intensive testing plans including their supervising.

6 Acknowledgments

Many thanks go to my colleagues Tilman Becker and Gerd Herzog for proofreading and their valuable comments and suggestions on earlier versions. The responsibility for the content lies with the author.

References

1. Galaxy Communicator Web Page, 2002. http://communicator.sourceforge.net.
2. Adam Cheyer and David Martin. The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, March 2001. OAA.
3. G. Herzog, H. Kirchmann, S. Merten, A. Ndiaye, P. Poller, and T. Becker. MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems. In *Proceedings of the HLT-NAACL03 Workshop on The Software Engineering and Architecture of Language Technology Systems (SEALTS)*, Edmonton, Canada, 2003.
4. G. Herzog, H. Kirchmann, S. Merten, A. Ndiaye, P. Poller, and T. Becker. Large-scale software integration for spoken language and multimodal dialog systems. In *Special issue on "Software Architecture for Language Engineering" of the Journal of Natural Language Engineering*, volume 10-3/4, pages 283–305, 2004.
5. David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, January-March 1999. OAA.

This article was processed using the LaTeX macro package with LLNCS style