From Business Process Models to Distributed Software Architecture

Volker Gruhn

University of Dortmund, Software Technology
Baroper Str. 301
44227 Dortmund, Germany
0049 (0) 231 755 2782

gruhn@ls10.informatik.uni-dortmund.de

Ursula Wellen

University of Dortmund, Software Technology
Baroper Str. 301
44227 Dortmund, Germany
0049 (0) 231 755 47 48

wellen@ls10.informatik.uni-dortmund.de

1. MOTIVATION

While examining complex business processes we have discovered that most of them are software-driven, i.e. most business processes depend on software systems. One approach to specifying these software systems is the development of business process models and the derivation of requirements for the software systems supporting them. In other words: business process models represent some sort of requirements for software systems needed to support the processes [1].

A closer look at the world of business process engineering [2] shows that more and more business processes are distributed [3,4]. Insurance companies for example have their main headquarters, many offices and vast numbers of agents. They all need access to central software systems, and they have their own software, too. Despite this distribution, couplings between dispersed systems are in their infancy [5]. Interfaces based on EDI or even based on a distributed software are frequently missing [6]. Instead of that, insurance agents have to fill out application forms either manually or within their local software system. They have to send the forms by letter post to the office where the data will be copied again manually into the central software. Here there are special software routines for verifying the application data and for the decision whether an application will be accepted or not.

It seems that the coupling of distributed process parts would be promising for different areas of business fields because it helps to speed up distributed processing, to save costs and to ensure high transparency. But these potential benefits can only be exploited, if the supporting software reflects distribution issues appropriately. In the following, to verify this idea, we consider a business process model which depicts the process of applying a life insurance policy.

2. MODELS OF DISTRIBUTED BUSINESS PROCESSES

A “global process model” shows how a process model is distributed. It abstracts details of local process models. As an example we refer to figure 1.

Figure 1 shows the model of the distributed process for applying a life insurance policy with the graphical notation of FUNSOFT nets [7,8] (rectangles represent activities, called agencies, circles represent information stores, called channels). The FUNSOFT net of figure 1 represents a global process which covers three types of site. The activities within the left circle are carried out by an insurance agent, the activities within the middle circle depict the office site, and the activities in the right circle are carried out in the headquarters.

These three types of sites (agent site, insurance office, insurance headquarters) are connected by certain types of communication channels. Communication channels have attributes like data volume, type of communication (push or pull communication), conversion function needed, and cardinality (number of sites participating in this communication). Especially for the attribute “cardinality” it is obvious that the number of participating sites has an enormous influence for the corresponding software architecture. If there is more than one participant at one end of the communication channel, a multi-user functionality for the connected software module is necessary. Details about communication channels and about their graphical representations are discussed in section 3.

Figure 1: Global process model of the application for a life insurance

The process model of figure 1 describes the overall process of applying for a life insurance. In co-operation with the customer the agent fills in an application form. After completing the form it is sent to the insurance office for recording, assigning an insurance no. and for checking the completeness of the data. The next process step is to send the electronic data to the headquarters for a risk analysis. This analysis leads either to a rejection of the application form or the customer will be insured. In the latter case the policy will be sent to the agent, to get a detailed profile of the customer. The headquarters forwards this policy to the insurance agent who delivers it to the customer.

The three clusters depict the three local process models (headquarters, office, agent) at a very abstract level. For our purpose this level is sufficient, since we focus on architectural distribution issues in the following.

Software supporting this distributed process has to satisfy the functional requirements defined by the activities of the process model and it has to fulfill the distribution requirements in the global model. From a distribution point of view, it may be very useful to develop an electronic interface between the agent's application software and the policy preparation software because this could speed up the communication between distributed process parts.

3. DERIVING DISTRIBUTED SOFTWARE ARCHITECTURE

In implementing a distributed business process in the insurance industry we recognized that the difficulty is to take the existing legacy systems and break them into pieces such that they can be distributed over the sites involved. In most cases there are several ways to do so. These

ways cover central software systems with local access, client/server solutions and they cover Internet-based solutions with remote browsers used as interfaces to otherwise central systems. The question which kind of distribution is most appropriate in a given situation is affected by two main criteria:

- Costs for telecommunication needed, the distribution of these costs to partners involved and the risk related to temporal unavailable telecommunication infrastructure. With respect to these criteria the communication channel parameters (like volume of data, frequency of data transfer) help a lot in conveying how a distributed process is supposed to behave and what telecommunication costs will occur.
- Effort needed to implement interfaces between remote software components. This effort is determined by the structure and documentation of the software which is already in place.

It turned out that our precise specification in terms of local process models and their interfaces is an important help in deciding about interface implementations. For the insurance process discussed in figure 1, we implemented a distributed software system which exchanges objects of type "offer" in an EDIFACT format and which ensures the exchange of policies by allowing remote, browser-based access to the database of the headquarters.

Based on the implementation of this process and on some implementations from the area of apartment administration [9], we developed certain components which improve the implementation of communication services. Even more important, it turned out that the architecture of

communication components seems to show common patterns. Which of these patterns are useful in a certain situation depends on very few parameters which determine the type of communication. These communication patterns implement certain exchange formats and certain types of synchronous and asynchronous message transfer. These predefined components can be used in the context of various process implementations. Since our experience is that these components usually have to be customized, they are organized as a framework of components which allows white-box reuse. That means, the components cannot only be reused as they are, but they can be used as supertypes from which context-specific subtypes can be derived.

If we refer to the interfaces between the insurance headquarters and the insurance offices as discussed in the previous section, then we recognize that the parameters defining the communication look as follows:

The interface is needed to exchange data of type “insurance customer” once a day. It is assumed that an object of type “insurance customer” occupies 250 byte and that ten objects of this type are exchanged each day (on average). Moreover, it is assumed that the insurance office actively sends this information (push communication) and that it expects an acknowledgement of receipt within two hours after sending the data.

Additionally, the sender expects to be informed when an insurance contract is issued, because it waits for this information to start a provisioning request.

This characterization of the interface has an impact on the software supporting this interface. The architecture of this software is displayed in figure 2.

Figure 2 shows three main subsystems involved: the interface software running at the insurance headquarters, the interface software running at the insurance offices and the communication component itself (which may be running at the headquarters or at the insurance office site). These three subsystems are implemented as operating system processes exchanging information by means of a middleware component. While the headquarters and the office component are basically determined by the functionality required by the insurance business (partially expressed by the process models, compare to figure 1), the communication component (central part of figure 2) is more general. It ensures that the underlying middleware component behaves in the specified way, for example as a persistent message queue or as an active data transfer agent. Depending on the concrete attributes of the communication channel, certain optional relationships indicated by dashed lines come to life.

The behavior of the communication component depends on further attributes of a communication channel. While details of communication channels are specified by means of attributes, the basic type of a communication channel is represented by graphical symbols. Figure 3 shows a subset of these symbols (compare also to figure 1). A type I communication channel indicates that the information to be exchanged is persistently stored at the sender site, a type II communication channel means that information is stored at the sender site and that the communication is based on a pull mechanism. The table in figure 3 sums up the meaning of the different types of communication channels.

Figure 2: Architecture of software needed for supporting the communication between insurance office and insurance headquarters

Legend: —→ Call relationship, - - - - -→ service will be realized with (optional)

	Persistency at site A	Persistency at site B	Communication
I	Yes	No	Push
II	Yes	No	Pull
III	No	Yes	Push
IV	No	No	Push

Figure 3: Communication modes between local process models

Depending on the concrete type of communication it may have a separate database or it may be based on a message-oriented middleware component (indicated by the box annotated with storage facilities). The concrete behavior of the communication component depends on the parameters describing the interface. It can realize a push mechanism (in this case it accesses the insurance office component “contract” first in order to find out if there are any messages which have to be pushed) or a pull mechanism (in this case the “communication manager” components access the “contract” component in order to identify whether or not contract messages have to be transferred).

Another typical element of the architecture is determined by the transformation of information from one site to another. For that purpose, the “convert” service of the communication manager accesses both formats (the one produced by the sender and the other expected by the receiver). Since most communication requests are based on standards for information exchange (EDIFACT, subsets of EDIFCAT, ISO standards) the “communication manager” accesses an information exchange framework. Based on this framework, certain conversion procedures are predefined. The general idea of the communication manager is to first translate an incoming message into an internal format and convert this internal format into an external format. Even though this two-step conversion involves more effort, it turned out to be useful in increasing the potential for reuse

and in using the middleware components as they are. In other words, the storage facilities (let alone if a database or a message is used) remain unchanged, since both are used for the internal formats of messages.

4. CONCLUSION

Our future research is focused in two directions. Firstly, we are going to investigate to which extent specifications of distributed processes can be used for the generation of software architectures needed to implement the processes. Secondly, we are continuously trying to improve our framework of communication components. Already, this framework substantially eases the implementation of distributed processes and we believe that further experience will yield in communication patterns which will provide support for certain types of process distribution.

5. REFERENCES

- [1] B. Warboys. Reflections on the Relationship Between BPR and Software Process Modelling in: Proceedings of the 13th International Conference on the Entity-Relationship Approach, December 1994.
- [2] M. Hammer, J. Champy. Reengineering the Corporation. Publisher: Harper Business, 1993.
- [3] I. Alloui, S. Latrous, F. Oquendo. A Multi-Agent Approach for Modelling, Enacting and Evolving Distributed Cooperative Software Processes in: Software Process Technology – Proceedings of the 5th European Workshop, October 1996, Springer.
- [4] G. Graw, V. Gruhn. Distributed Modeling and Distributed Enaction of Business Processes in: Software Engineering – ESEC95, 5th European Software Engineering Conference, September 1995, Springer.
- [5] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. Specifying Distributed Software Architectures in: Proceedings of the 5th European Software Engineering Conference, September 1995.
- [6] M. Merz, K. Müller, W. Lamersdorf. Service trading and mediation in distributed computing environments in: Proceedings of the 14th International Conference on Distributed Computing Systems, publisher: IEEE Computer Society Press, 1994.
- [7] W. Deiters, V. Gruhn Managing Software Process in MELMAC in: SDE4, December 1990.
- [8] V. Gruhn, J. Urbainczyk Software process modeling and enactment: an experience report in: Proceeding of the 20th International Conference on Software Engineering, April 1998.
- [9] V. Gruhn, S. Wolf Software Process Improvement by Business Process Orientation in: SPIP, August 1995.