Tool-Based Business Process Modeling using the SOM Approach

Abstract

Business processes play an important role in analyzing and designing a company's behaviour and organization. Modeling business processes is an integral part of the Semantic Object Model (SOM). The paper shows the basic concepts of business process modeling using the SOM approach and presents the design goals and architecture of an accompanying tool.

1. Basic Concepts of Business Process Modeling Using the SOM Approach

The Semantic Object Model (SOM) is a comprehensive approach for analysis and design of business information systems and specification of business application systems (see [FeSi93a] and [FeSi93b]). The scope of the whole approach covers three major steps:

- **Enterprise planning**: Identification of a company's universe of discourse, its goals and objectives[2], its success factors as well as its value chains.
- **Business process modeling**: From a behavioural viewpoint, a company consists of a set of business processes (see [FeSi93c]). Main processes contribute directly to the company's goals, sub-processes support the main processes with some kind of services. Relationships between business processes follow the client-server model. A client process engages a server process to deliver a certain service. SOM provides an approach for business process modeling using a semi-formal, object-oriented notation.
- **Specification of business application systems**: The purpose of a business application system is to automate some part of a business process. Application systems are identified and separated within the set of business processes and are specified using an object-oriented notation.

This section of the paper concentrates on business process modeling based on the meta model shown in figure 1.

1 {Univ.-Prof. Dr. Otto K. Ferstl, Univ.-Prof. Dr. Elmar J. Sinz, Dr. Michael Amberg, Dipl.-Inf. Udo Hagemann, Dipl.-Inf. Carsten Malischewski}, Business Informatics, University of Bamberg, D-96045 Bamberg, Germany. Phone ++49 951 863 {2679,2512,2578,2540,2516}, Fax ++49 951 39636, Internet {ferstl, sinz, amberg, hagemann, malischewski} @sowi.uni-bamberg.d400.de

2 We use the term *goal* to denote the intended final state of an object, achieved by the execution of a task. The term *objective* refers to the corresponding quality aspects, aimed during the execution of the task.

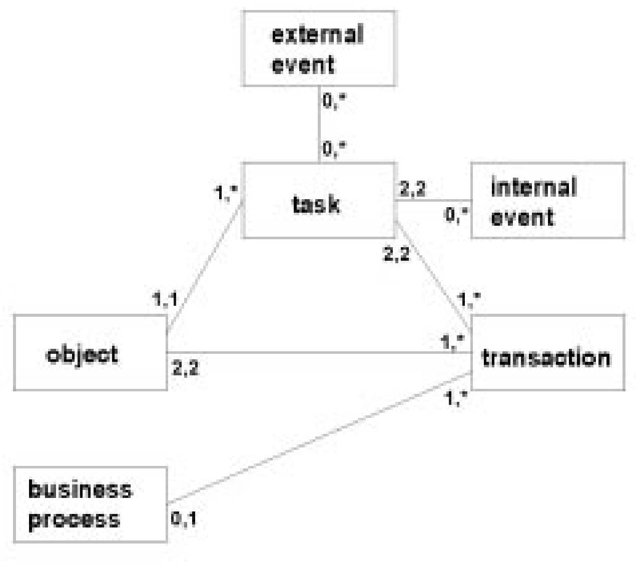

Fig. 1: Meta model of the SOM approach for business process modeling

Business processes consist of business **transactions**. From a static viewpoint, each transaction builds a communication channel between two **objects**. Referring to the delimitation of the universe of discourse **internal objects** and **external objects** are distinguished. From a dynamic viewpoint, a transaction controls and executes the exchange of services and messages respectively. Each object is associated with a set of **tasks**. A task can be regarded as an operator of an object which drives a transaction. Each transaction is driven by exactly two tasks belonging to different objects. **Internal events** can be used to define sequences of tasks within an object. **External events** define environmental pre-conditions for the execution of tasks.

A business process described according to the meta model above can be refined recursively to a more detailed level. This is done by decomposing objects, tasks, and transactions as well. Decomposition of objects and transactions uncovers the basic coordination mechanisms between objects (see fig. 2):

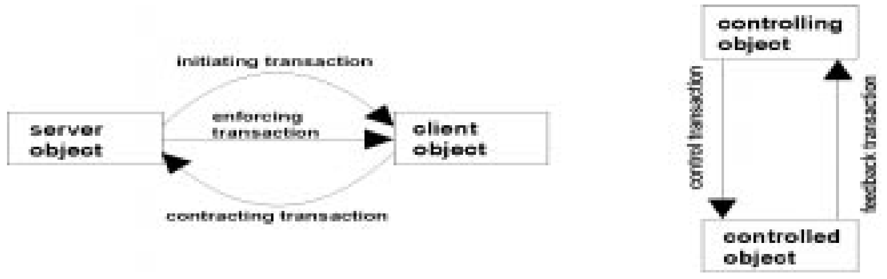

Fig. 2: Basic coordination mechanisms between objects

- **Negotiation principle**:

  A business transaction between two objects is decomposed into a sequence of sub-transactions: (1) an initiating transcation, (2) a contracting transaction and (3) an enforcing transaction. During the **initiating transaction**, the objects learn to know each other and exchange information on deliverable services. Within the **contracting transaction** both objects agree to a contract on the exchange of a service. The purpose of the **enforcing transcation** is to exchange the service between the objects.

- **Feedback control principle**:

  An object is decomposed into two sub-objects and two transactions which establish a feedback control loop. The controlling sub-object prescribes objectives or sends controlling messages to the controlled sub-object by a **control transaction**. A **feedback transaction** closes the feedback control loop by reporting to the controlling object.

2. Example for Business Process Modeling

In the following the business process *distribution* is examined as an example. At the initial level, this business process consists of three components: (1) an internal object *distributor*, (2) an external object *customer*, and (3) a transaction *delivery* which models the service delivery from distributor to customer.

The next level consists of a refinement of the initial level. First of all the *delivery* transaction is decomposed to uncover the coordination between *distributor* (server object) and *customer* (client object). As the two objects negotiate about the service delivery, the transaction is decomposed according to the negotiation principle into the sub-transactions *price list* (initiating), *order* (contracting), and *delivery* (enforcing transaction).

In the next step the object *distributor* is decomposed according to the feedback control principle. This leads to the sub-objects *sales* (controlling sub-object) and *store* (controlled sub-object). At the same time the sub-transactions *price list*, *order*, and *delivery* are re-assigned to the sub-objects. The *sales* sub-object deals with *price list* and *order*, the *store* sub-object has to manage the *delivery* transaction.

In addition, decomposition of the *distributor* object requires the sub-transactions *delivery order* (control transaction) and *delivery report* (feedback transaction) from *sales* to *store* and from *store* to *sales* respectively. The purpose of *delivery order* is to connect the *order* sub-transaction to the *delivery* sub-transaction.

Figure 3 shows the business process model at this level of refinement. The top left window shows sub-objects and sub-transactions forming the **interaction scheme** view on a business process model. The bottom right window shows the sequence of sub-transactions and the tasks to drive them. This graph is called the **process event scheme** view and follows the petri-net concept.

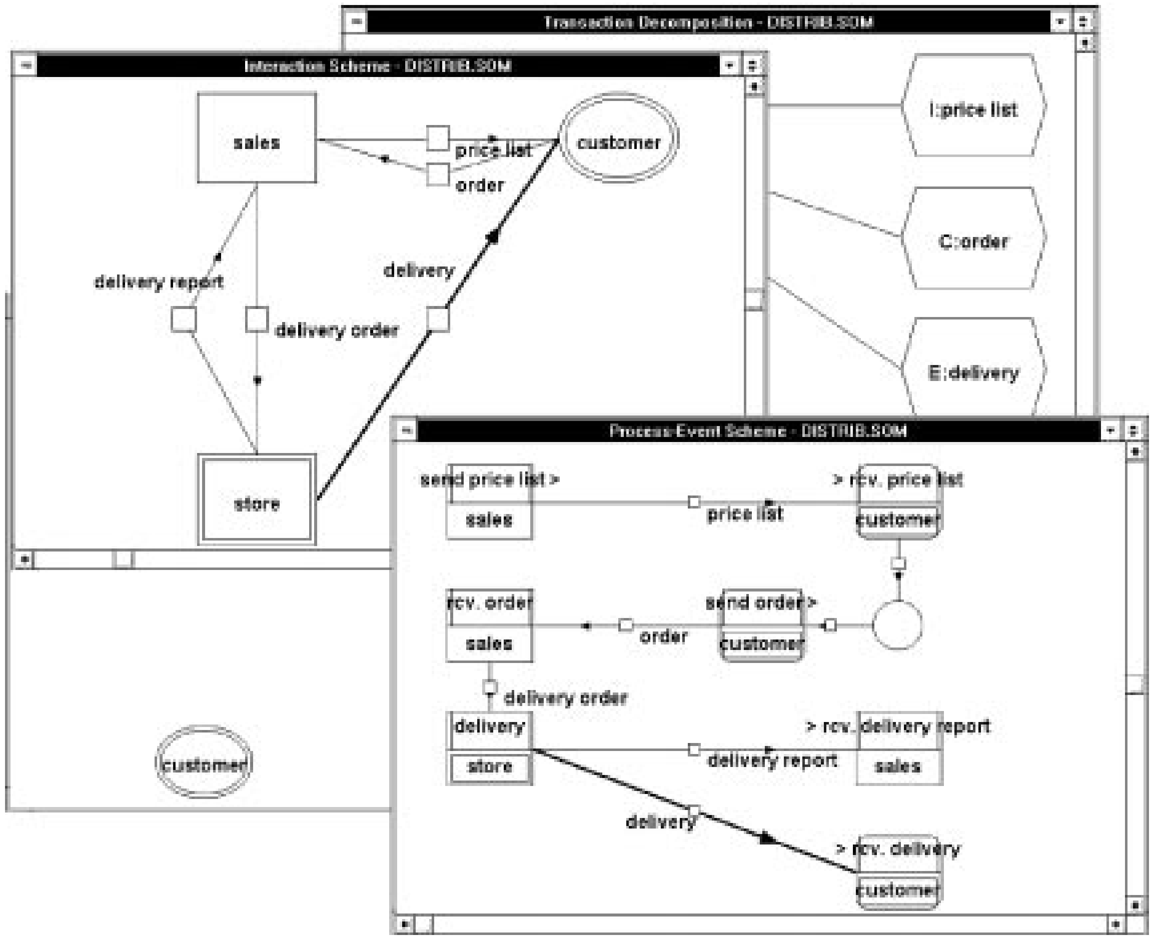

Fig. 3: Example: business process *distribution*; SOM tool view

3. Tool support of the modeling process

Design Goals

The meta model shown in figure 1 comprises all valid business process models, including the initial model to start with. In addition, transformation rules for the decomposition of transactions and objects restrict each design step that transforms a consistent business process model into a more detailed one (see [FeSi93b]). A business process engineer has to obey and must be assisted in obeying the meta model as well as the transformation rules. Consequently the SOM tool provides assistance in two dimensions:

1. **Modeling process:** Modeling starts with a minimal model of a business process, consisting of an internal and an external object and a transaction between them. The design process consists of a series of design steps which are restricted by the given set of transformation rules. Each design step automatically generates additional structures as needed. For example, decomposing a transaction automatically decomposes its driving tasks.

2. **Presentation of a model:** In analogy to technical drawings used for construction purposes, an engineer is provided with different views onto a business process model which present the modeling results in a convenient and effective way. Each view focuses on a different aspect and is available at a selectable granularity. All views together present a complete and sound model of a set of business processes.

Tool architecture

A tool architecture has to ensure the manageability of large modeling projects in several aspects. The project repository must be made accessible to multiple engineers using heterogenous hardware platforms. Every engineer has to be supported in handling the complexity of a model's representation as well as taking different points of view onto a model (see [FeHa92]). These requirements lead to a multi-view approach like the PAC model (presentation, abstraction, control; see [Cou87]) combined with a client-server architecture (fig.4).

The tool is implemented in an object-oriented technique as a set of reusable class libraries. As each meta model component is handled by a separate class, the tool's core class structure is identical to the meta model. The core classes have been extended to support the multi-view functionality, namely by the notion of a **local/global** visibility state and a current **focus**. The local visibility state affects a structure's visibility in exactly one view, whereas a structure's global state affects all its representations in several views. Manipulating a structure's visibility is used to reduce the complexity of a model's representation systematically. The focus concept helps to support navigation through a model. If a certain part of a schema is moved into the center of a view, the other views are adjusted automatically.

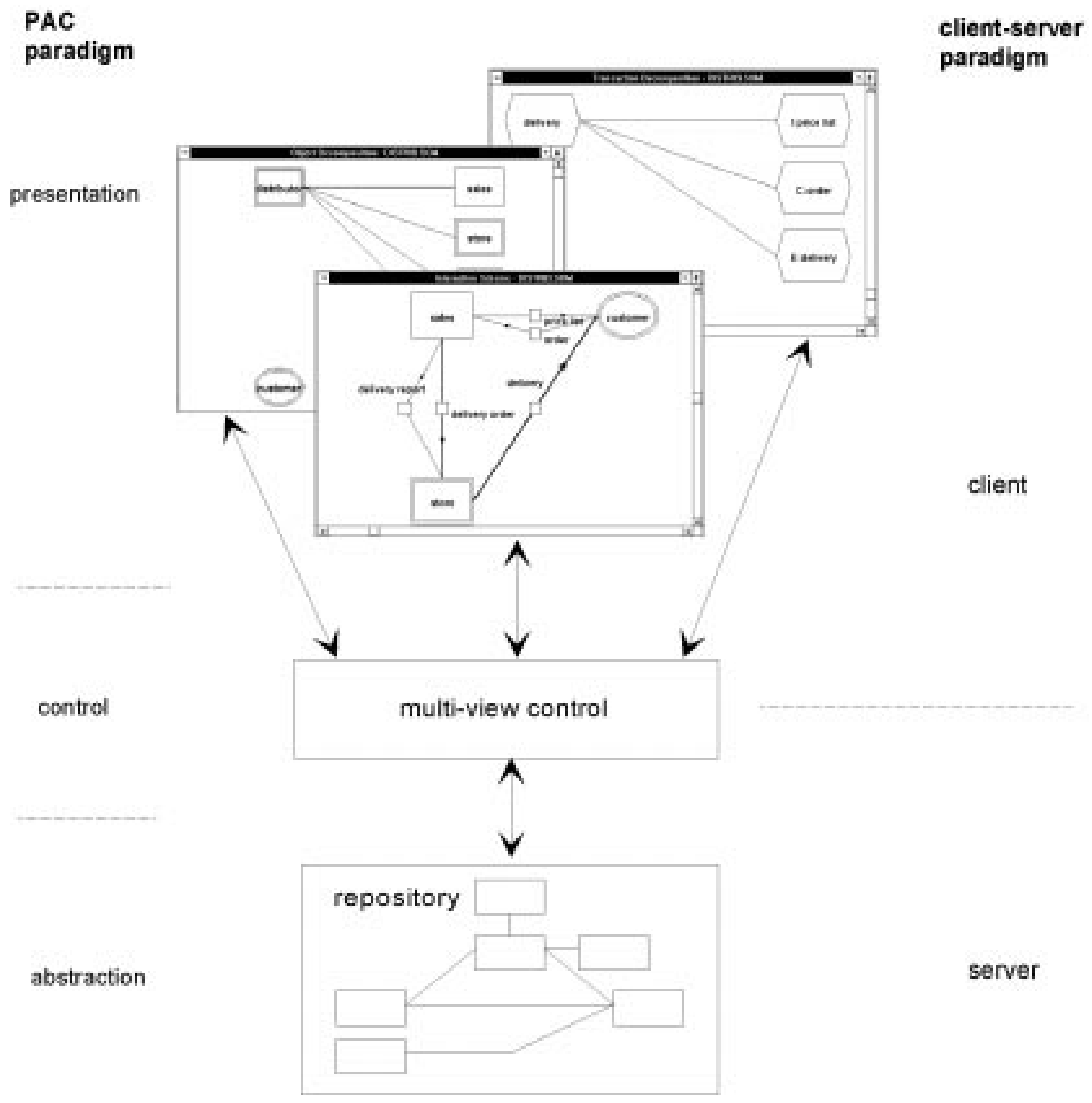

Fig. 4: Architecture of the SOM tool

User interaction via a graphical user interface is encapsulated by a separate class library. These libraries contain roughly 240 classes, 30 of them being root classes without ancestors. The inheritance structure forms a forest of classes with a maximum depth of 8. Using these libraries the task of building an application consists of writing a single application class that activates some views, possibly modifying some view's behaviour via subclassing, and linking it with the existing class libraries.

4. Tool availability and experiences

The portability of a tool is mainly affected by the programming language, the user-interface management system (UIMS), and the underlying database system. We decided for C++ as the currently most common object-oriented programming language. We use the user interface toolkit XVT to make the graphical programming interfaces independent of different hardware platforms. Using C++ and XVT, source code compatibility for MS-Windows, SUN OpenLook (SUN SPARC, SunOS 4.1.3), and OSF/Motif platforms (Iris Indigo, Irix 4.0.5f and SNI MX300i, Sinix 5.41) has been achieved. Data management is done on the basis of flat files until now. Currently we are preparing for the use of the object-oriented database system ObjectStore, which is available for the platforms mentioned above.

The tool is currently being used for educational purposes, in research projects, and in commercial projects as well.

References

[Cou87] Coutaz, J.: The Construction of User Interfaces and the Object Paradigm; In: Bezirin, J. et al (eds.) Proc. ECOOP '87, Paris, Springer, New York, 1987, 121-130

[FeHa92]* Ferstl, O.K.; Hagemann U.: Die Visualisierung der SOM-Diskurswelt in einem Multiview-Ansatz; Fachberichte Informatik 7-92, Universität Koblenz-Landau, 1992

[FeSi93a] Ferstl, O.K.; Sinz, E.J.: Grundlagen der Wirtschaftsinformatik; Oldenbourg, München, 1993

[FeSi93b]* Ferstl, O.K.; Sinz, E.J.: Der Modellierungsansatz des Semantischen Objektmodells (SOM); Bamberger Beiträge zur Wirtschaftsinformatik, Nr. 18, Bamberg, 1993

[FeSi93c]* Ferstl, O.K.; Sinz, E.J.: Geschäftsprozeßmodellierung; In Wirtschaftsinformatik 35(6), 1993, 589-592

- Documents marked with "*" can be retrieved via WWW from http://www.seda.sowi.uni-bamberg.de/lehrstuhl/publikationen/.

- This work is partly supported by the Deutsche Forschungsgemeinschaft (DFG) under contract No. Si 481/1-3.