Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -

Otto K. Ferstl, Elmar J. Sinz

Abstract

The paper presents a comprehensive and integrated approach for business process modeling and for the specification of distributed business application systems. The approach is based on a framework of business systems which consists of three layers: (1) business process model, (2) business application systems, and (3) computing systems. This framework is used to manage the complexity of a business system and to address the different actors and abstractions of actors within a business system (e.g. persons and machines, application systems, and computing systems). The linkage of the layer of business application systems to both the layers of business processes and computing systems helps to specify the distribution of business application systems in a natural way. Object-oriented concepts are used as universal enabling techniques.

Keywords: business system, business process, business application system, business object, business transaction, coordination mechanism, semantic object model (SOM)

Contents

1. Introduction

This paper presents a new approach to business process modeling and to the specification of distributed business application systems. The approach is based on a multi-layered framework which consists of three layers: (1) business processes, (2) corresponding business application systems, and (3) computing systems to execute the application systems. All three layers may be distributed, employing multiple cooperating actors. Actors are persons and machines (e.g. production machines, transport systems, assembly lines, and computers). The layer of the business application systems is explicitly linked to the layer of the business processes. The initial structure of the specification of a business application system can even be derived directly from the corresponding business process model.

Throughout the paper, we take a system-oriented viewpoint on *business systems* (e.g. companies, business units and departments). From an outside view, a business system is considered open and goal-oriented. The inside view of a business system is developed by a stepwise decomposition which uses two different criteria:

- Production and delivery relationships lead to a decomposition into cooperating business processes. The relationships between the business processes follow the client/server principle. In the role of a server, a business process produces and delivers its outcome to one or more client processes. Each business process can act in both roles.

- Control relationships lead to a decomposition into a controlled subsystem which produces and delivers the outcome of business processes, and a controlling subsystem which is called the business information system. It works as the *nervous system* of a business system (see Ferstl & Sinz (1994)). A business information system consists of automated and non-automated sub-systems. An automated sub-system is called a *business application system*; a non-automated sub-system is part of the personnel organization.

In the following chapters the approach is outlined in detail. Chapter 2 introduces the concept of distributed systems and specializes the concept into distributed business systems, distributed business application systems, and distributed computing systems according to the layers mentioned above. Chapter 3 explains our approach to business process modeling especially focusing on the coordination principles within business processes. Based on the architecture of distributed business systems we develop in chapter 4 an architecture of distributed business application systems and link both layers. In this way we are able to provide a unified approach to business process modeling as well as to the specification of distributed business application systems.

2. Distributed Systems

In general, a *distributed system* is characterized by the following properties (see also Enslow (1978)):

a1) From an outside view, the system is a black box and pursues a set of joint goals.

b1) The system consists of multiple autonomous components which cooperate in pursuing these goals. There is no component which has global control of the system.

Property (a1) requires that a distributed system be an *integrated system*, consisting of interacting components. Isolated components do not pursue joint goals. Property (b1) refers to the

autonomy of the cooperating components. This means that the components generate separate concurrent processes while executing their tasks. Concurrent processes communicate by exchanging messages and service packages.

The definition of a distributed system is applicable to business systems, business application systems, and computing systems. For example, a *distributed business system* consists of multiple interacting business processes, each of them pursuing joint enterprise goals.

In the case of *distributed business application systems* and *distributed computing systems*, the definition given above can be extended:

a2) The distribution of the system into multiple components is invisible to any user of the system. This feature is called *system transparency.*

b2) The components of a distributed system are loosely coupled. Each component is autonomous and encapsulates both structure and behavior.

Figure 1 shows the difference between loosely coupling and tightly coupling of components. While *loosely coupled components* have their own memory and communicate by exchanging messages via a communication channel, *tightly coupled components* communicate by sharing a common memory (see Gray & Reuter (1993) and Martin, Pedersen & Bedford-Roberts (1991)).

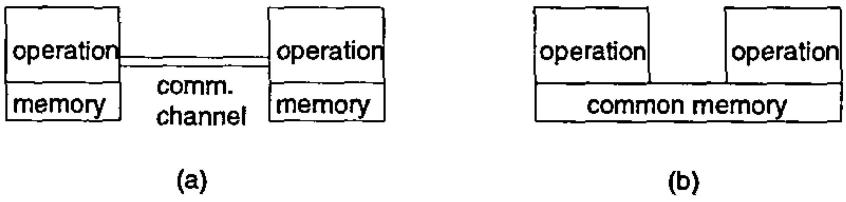


Fig. 1: Loosely coupled (a) and tightly coupled components (b)

A system can have different degrees of distribution. Shifting the coupling of components from loose to tight decreases the system's degree of distribution. On the other hand, splitting tightly coupled components into loosely coupled components increases the system's degree of distribution.

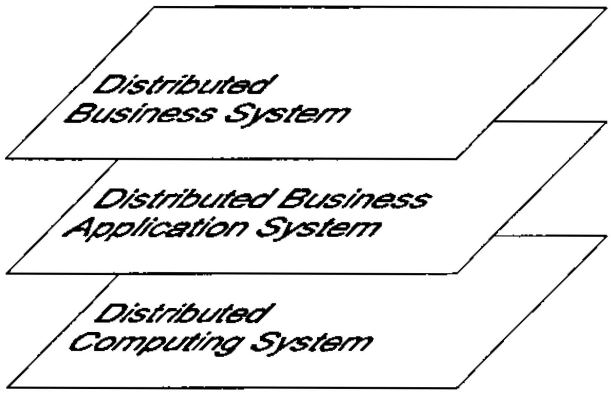


Fig. 2: Conceptual framework for distributed systems

Throughout this paper the specification of distributed systems follows the conceptual framework shown in figure 2. It distinguishes three layers of specification:

- 1st layer: Specification of a *distributed business system* as a set of cooperating business processes. Each business process can be decomposed into loosely coupled subprocesses, thereby becoming distributed.

The purpose of a business application system is to automate or to semi-automate tasks of a business process. The whole business process may be supported by multiple interacting application systems. Identifying the scope of these business application systems within a business process model leads to the second layer of specification.

- 2nd layer: Specification of a *distributed business application system.* At this layer, the domain-related components of an application system are divided into sub-components for communication, application, and data management. Assuming that all sub-components at this layer are loosely coupled leads to the domain-related specification of a business application system with a maximum degree of distribution (see fig. 9).

Starting with a domain-related specification with a maximum degree of distribution, the coupling of the components can be shifted from loose to tight according to specific design goals. This change leads to a domain-related specification of a business application system with a convenient degree of distribution. Finally, the resulting components utilize the virtual machines specified at the third layer.

- 3rd layer: Specification of a *distributed computing system*, consisting of virtual machines. These machines are actors for the components of the distributed business application system. Examples of virtual machines are operating systems, database management systems, user-interface management systems, and application-independent class libraries. They run on computers and communication networks.

Object-oriented techniques promise to be suitable for the specification of distributed systems on all three layers. This is because these techniques specify loosely coupled, autonomous components in a natural way. This paper concentrates on the object-oriented specification of the first and second layer as well as the transition between these layers. The third layer is conceptually integrated, but not outlined in this paper.

3. Business Process Modeling

There are various meanings of the term business process in literature and practice (see Ferstl & Sinz 1993b). The definition of business processes used here follows the approach of the *Semantic Object Model (SOM)* (see Ferstl & Sinz (1990), Ferstl & Sinz (1991), Ferstl & Sinz (1993a), and Ferstl, Sinz, Amberg, Hagemann & Malischewski (1994)). Within the SOM approach, business process modeling is embedded as part of a comprehensive analyis and design method for business systems. The SOM approach covers three major steps:

- *Enterprise planning*: Identification of a company's universe of discourse, its environment, its services, its goals and objectives[129], its success factors, and its value chains.
- *Business process modeling*: Identification of main processes and service processes. Main processes contribute directly to the company's goals, service processes provide their outcome to main processes or other service processes. Relationships between business processes follow the client/server model. A client process engages a service process to deliver a certain service.
- *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.

3.1. Meta Model for Business Process Modeling

This section of the paper concentrates on business process modeling based on the meta model shown in figure 3. The purpose of a business process is to produce different kinds of services and to transmit the services to the clients of the business process. Therefore, the specification of a *business process* consists of the following elements:

[129] We use the term *goal* to denote the intended final state of an object, pursued by the execution of a task. The term *objective* refers to the corresponding quality aspects, aimed by the execution of the task.

- Specification of the *services* the business process deals with.
- Specification of a *business object* and a set of *business transactions* which produce and transmit the services. Concerning the delimitation of the universe of discourse, *internal objects* and *external object* are distinguished. From a static viewpoint, a transaction builds a communication channel between two business objects. From a dynamic viewpoint, a transaction controls and executes the exchange of service packages and messages, respectively.
- Each transaction is associated with exactly two *tasks*. These tasks can be regarded as drivers of a transaction. One task provides and pushes a particular service package or message, and the corresponding task pulls it. The tasks carry out a protocol which is necessary to transmit a service package or a message from a server to a client.

  The tasks of a transaction are assigned to the business objects which are connected by the transaction. In this way, an object is associated with a set of tasks, each task driving a particular transaction. From the viewpoint of a business object, tasks can be regarded as the object's operations.

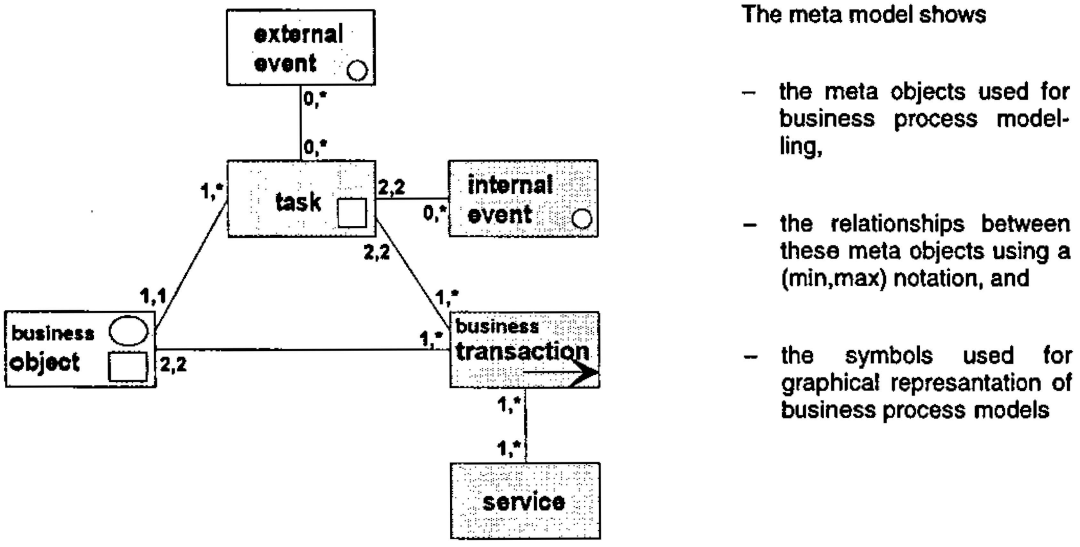


Fig. 3: Meta model of business process modeling

- In addition to the transactions, two kinds of *events* are used to control the execution of tasks. *Internal events* connect tasks within an object. *External events* define environmental pre-conditions for the execution of tasks.

Business processes are too complex to be presented in a unitary form. Therefore, two different views on business processes are used for representation. These views can be derived from the meta model.

- The first view is called the *interaction schema* and focuses on the static structure of a business process. It contains objects and transactions in their roles as communication channels.
- The second view is called the *task-event schema*[130] and presents the dynamic behavior of a business process. It consists of tasks, internal events, external events, and transaction-bound events.

3.2. Coordination of Business Objects

A business process described according to the meta model above can be refined recursively to a more detailed level. This is done by decomposing business objects as well as business transactions. Decomposition of objects and transactions uncovers the basic coordination mechanisms between objects (see fig. 4):

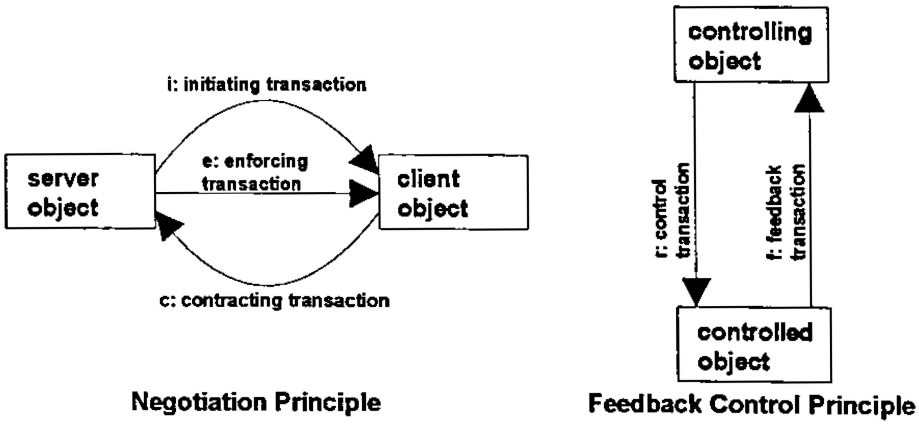


Fig. 4: Basic coordination mechanisms between objects

- *Negotiation principle*: Applying the negotiation principle, a business transaction between two objects is decomposed into a sequence of sub-transactions: (1) an initiating transaction, (2) a contracting transaction, and (3) an enforcing transaction. During the execution of an *initiating transaction*, the objects learn to know each other and exchange information on deliverable services. Within the *contracting transaction* both

[130] Task-event schemata are based on the petri-net concept and can be formally translated into petri-nets.

objects agree to a contract on the exchange of a service. The purpose of the *enforcing transaction* is to exchange the service between the objects.

- *Feedback control principle*: Applying the feedback control principle, an object is decomposed into two sub-objects and two transactions, together establishing 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.

3.3. Example: The Business Process Distribution

In the following, the business process *distribution* is examined to illustrate the coordination of business objects. 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 *service* which models the service delivery from distributor to customer. Figure 5 shows the interaction schema and the task-event schema belonging to this level of detail.

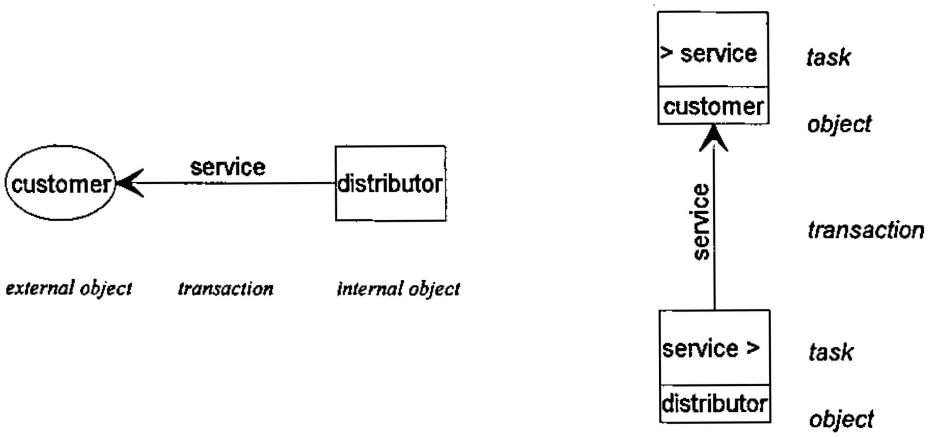


Fig. 5: Interaction schema (left) and task-event schema (right) of business process *distribution* (1st level)

In the task-event schema, the initial task names are derived from the transaction name. For example, the task names *service>* (say "send service") and *>service* (say "receive service") are derived from *service.*

The second level consists of a refinement of the initial level. Here the *service* transaction is decomposed to uncover the coordination between *distributor* (server object) and *customer* (client object). As the two objects negotiate about the delivery of a service, the transaction is decomposed according to the negotiation principle into the sub-transactions *i: price list* (initiating), *c: order* (contracting), and *e: service* (enforcing transaction). Because the sub-transactions are executed sequentially, the task-event schema corresponding to the interaction schema is determined implicitly (see fig. 6).

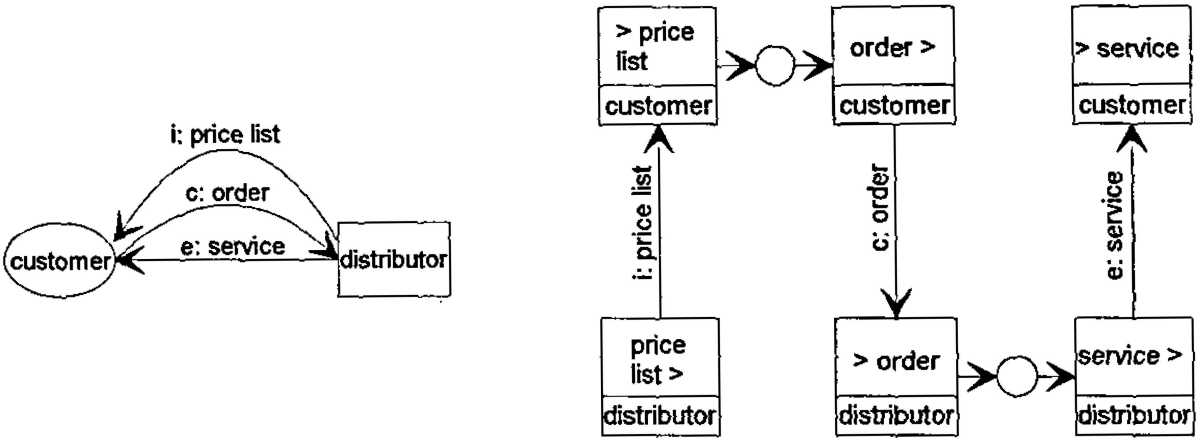
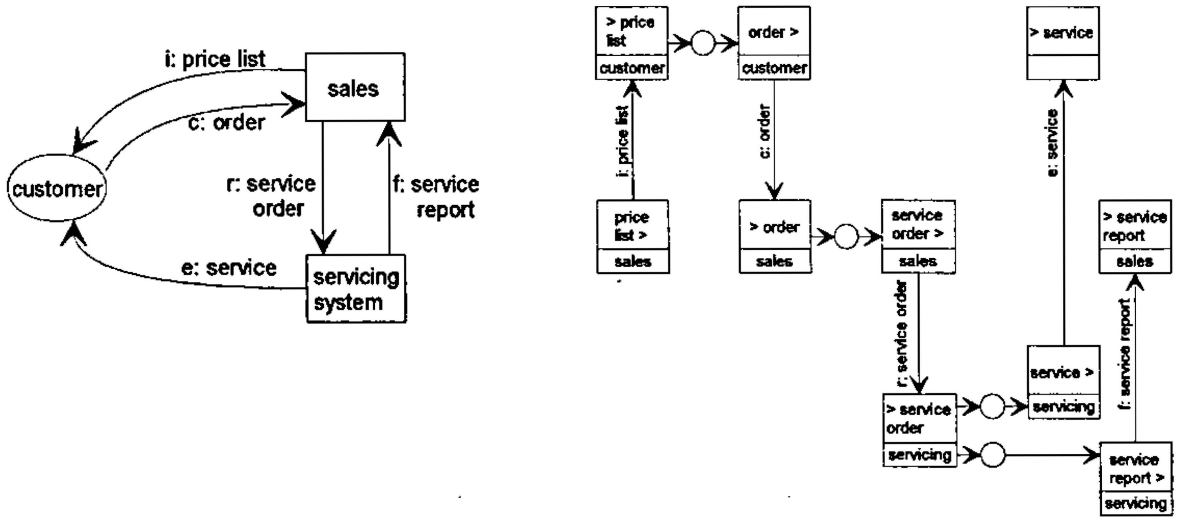


Fig. 6: Interaction schema (left) and task-event schema (right) of business process distribution (2nd level)

At the third level, the object *distributor* is decomposed according to the feedback control principle. This leads to the sub-objects *sales* (controlling sub-object) and *servicing system* (controlled sub-object). At the same time the sub-transactions *price list*, *order*, and *service* are re-assigned to the sub-objects. The *sales* sub-object deals with *price list* and *order*; the *servicing system* sub-object manages the *service* transaction.

In addition, the decomposition of the *distributor* object introduces the sub-transactions *r: service order* (control transaction) and *f: service report* (feedback transaction) from *sales* to *servicing system* and from *servicing system* to *sales* respectively. The purpose of *service order* is to connect the *order* sub-transaction to the *service* sub-transaction. Since the sequence of tasks cannot be fully derived from the interaction schema at this level of detail, the task-event schema shows additional information (see fig. 7).

Fig. 7: Interaction schema (left) and task-event schema (right) of business process *distribution* (3rd level)



The next decomposition leads to the fourth level (see fig. 8). First, the *e: service* transaction is decomposed into the sequence *e: delivery* and *e: cash up*. The *cash up* transaction is decomposed again according to the negotiation principle into the sequence *c: invoice* and *e: payment*. An initiating transaction is omitted here because the business objects already know each other. The contract of the *invoice* transaction refers to the amount of payment, not to the obligation to pay in principle.

As a result of this refinement, several other decompositions become necessary. The business object *servicing system* is decomposed into *store* and *finances*, responsible for products and payments respectively. *r: service order* is decomposed into the parallel transactions *r: delivery order* and *r: debit*. On the other hand, *f: service report* is decomposed into the parallel transactions *f: delivery report* and *f: payment report*.

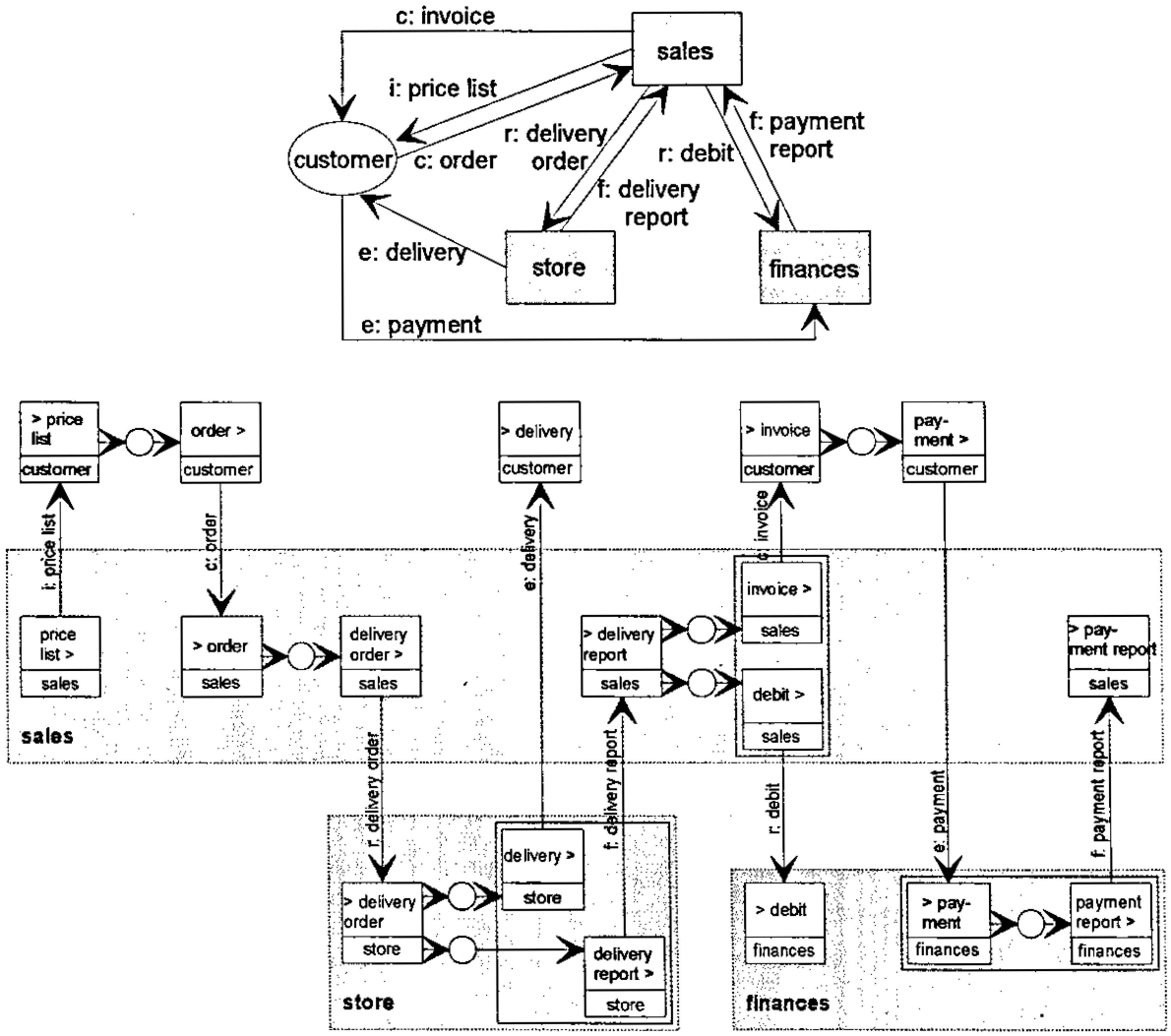


Fig. 8: Interaction schema (above) and task-event schema (below) of business process *distribution* (4th level)

4. Object-Oriented Specification of Distributed Business Application Systems Based on Business Process Models

A business process model as developed in the previous chapter provides a solid platform for an object-oriented specification of distributed business application systems. The corresponding conceptual framework, which is part of the SOM approach, has been outlined in chapter 2 and will be detailed below (see fig. 9).

Following this approach, business systems, business application systems, and computing systems are specified at the corresponding layer as distributed systems with a specific client/server architecture at each layer:

- Within a distributed business system, servers deliver clients with requested services.
- A distributed business application system consists of domain-related components for application, data management, and communication and client/server relationships between these components.
- A distributed computing system provides virtual machines which are connected by communication channels. These virtual machines may also establish a client/server architecture.

As proposed throughout this paper, the specification of a distributed business application system consists of a sequence of design steps, each step corresponding to one layer of the following conceptual framework:

1. Identifying the business application system within a business process model,
2. specifying the domain-related architecture of the distributed business application system according to the application/data management/communication concept, and
3. specifying the physical architecture of the distributed business application system which consists of computers and communication networks.

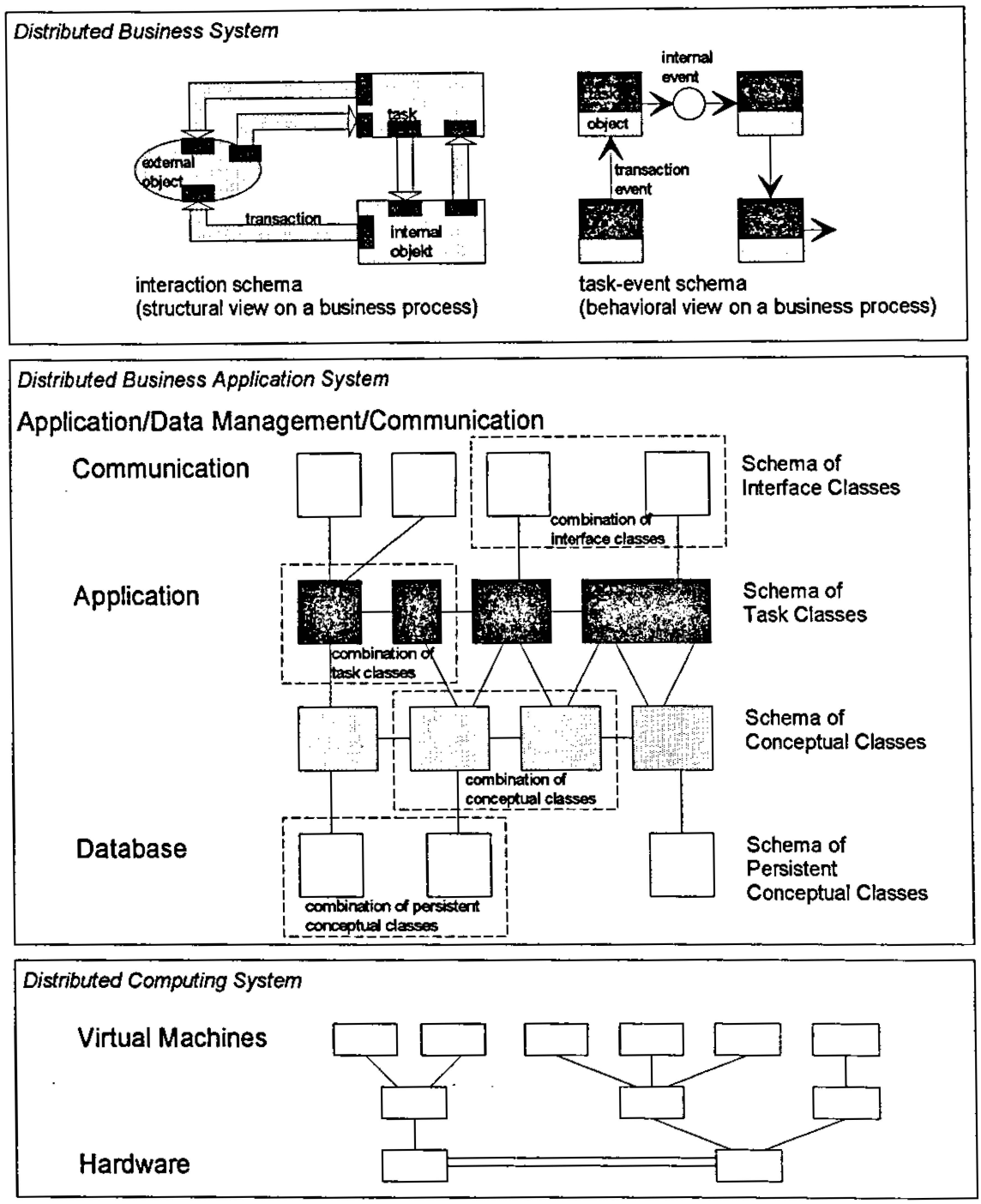


Fig. 9: Conceptual framework for the specification of distributed business application systems

The design of a distributed business application system is influenced by various design goals. Each design step focuses on specific design goals. Since all the goals are mutually dependent, the design of a distributed business application system constitutes a simultaneous decision problem, which is resolved through a sequence of design steps. The design steps and the associated design goals are explained in the following sections.

4.1. Identification of a Business Application System within a Business Process Model

The first layer of the conceptual framework shown in figures 2 and 9 specifies a distributed business system as a set of business processes. The interactions between these business processes constitute the *client/server architecture of the business system*. When a business process is decomposed into its sub-processes, the interactions of these sub-processes follow the client/server concept in turn.

A business application system automates or semi-automates some part of a business process. Therefore, the first design step of specifying a distributed business application system is to identify the scope of the application system within the interaction schema and the task-event schema of a business process model.

The design goal of this design step is to support business objects with *joint goals and objectives* by one application system. Following this guideline, the business process *distribution* should be supported by three application systems:

- a *sales* application system,
- a *store* application system, and
- a *finances* application system.

The scope of these application systems is shaded within the interaction schema and the task-event schema of the detailed business process model (see fig. 8).

4.2. Specification of the Domain-Related Architecture of a Distributed Business Application System

The second layer of figures 2 and 9 specifies the domain-related architecture of a distributed business application system. At this layer, an application system is decomposed into its domain-related components according to the application/data management/communication architecture.

Using the SOM approach, the application/data management/communication architecture of a distributed application system is specified by four schemata:

- The *application* sub-system is specified by the schema of conceptual classes and the schema of task classes. Conceptual classes establish the domain-related basis of a business application system. Task classes specify the cooperation of conceptual classes when executing a task.

- The *communication* sub-system is specified by the schema of interface classes.
- The *data management* sub-system is specified by the schema of persistent conceptual classes.

The *schema of conceptual classes* consists of specifications of classes with attributes and operations (see fig. 10). The classes are connected by *is_a*, *interacts_with*, and *is_part_of* relationships (see Ferstl & Sinz (1990)).

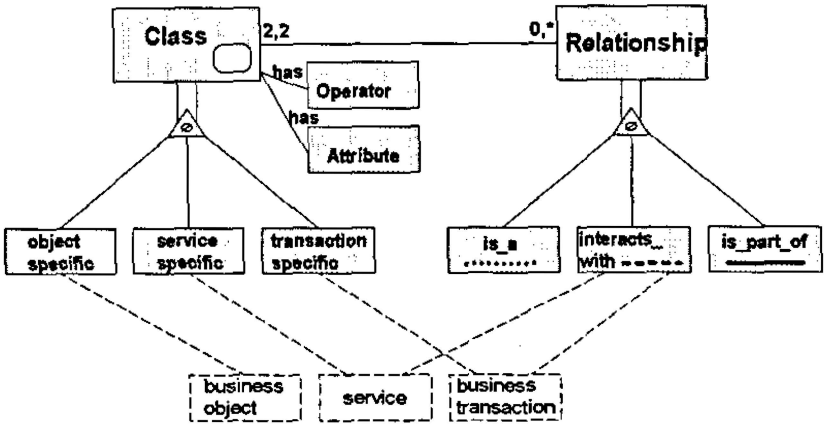


Fig. 10: Meta model of the schema of conceptual classes

According to the meta model shown in figure 10, an initial schema of the conceptual classes can be derived using the interaction schema and the task-event schema of the business process *distribution* at the most detailed level (see figure 8). Dashed rectangles and lines show the relationships between a business process model and a schema of conceptual classes. Business objects, service specifications, and business transactions lead to classes and *interacts_with* relationships. *is_a* and *is_part_of* relationships cannot be derived from the business process model at this stage. Existence dependencies between the instances of the classes are visualized from left to right by graphical placement of the class symbols. The resulting initial schema of conceptual classes is shown in figure 11.

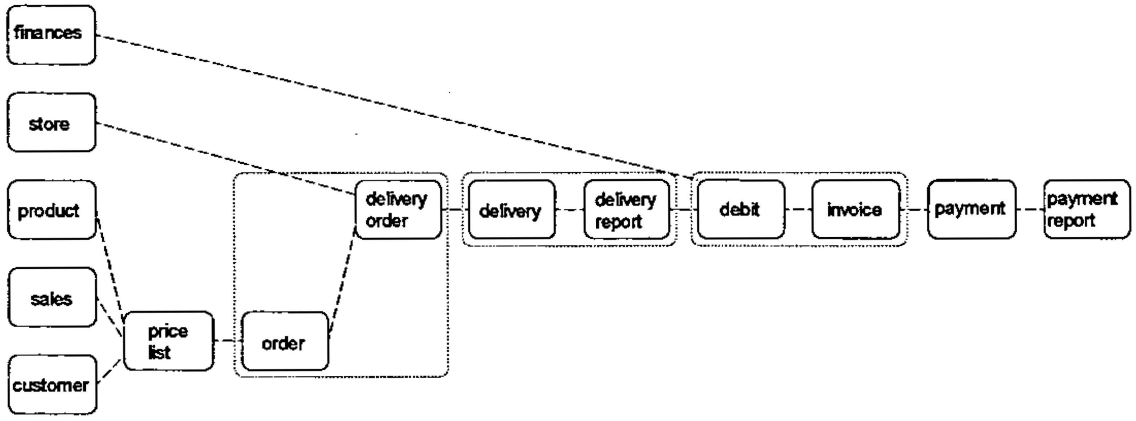


Fig. 11: Conceptual class schema derived from business process *distribution*

The schema of conceptual classes has to be completed by assigning attributes and operations to the class definitions, finding *is_a* relationships between classes, decomposing class definitions using *is_part_of* relationships and specifying the complexity of *interacts_with* relationships. These aspects are beyond the scope of this paper (see Ferstl & Sinz (1990)).

The *schema of task classes* specifies for each task the cooperation of conceptual classes during the execution of a task. The initial schema of task classes can be derived from the task-event schema of the business process model. Each task within the scope of the business application system leads in general to a separate task class (see fig. 8). Details of the specification of task classes are not explained here (see Ferstl & Sinz (1990)).

The *schema of interface classes* specifies interfaces for human-computer communication and computer-computer communication. Human-computer interfaces are used to implement semi-automated tasks, computer-computer interfaces are used to connect different application systems.

The *schema of persistent conceptual classes* specifies classes whose instances are persistent even when the execution of a task has been finished. Therefore, these instances have to be stored in a database. The attributes of the schema of persistent conceptual classes result from a projection onto the attributes of the schema of conceptual classes. In addition, each class specifies access operations onto these attributes.

The initial schemata as constructed above constitute the basis for the design step at the second layer of the conceptual framework. The design goal of this step is to remove *domain-related redundancy and inconsistency*. This is done by combining particular classes within the initial schemata. Combining two classes shifts their coupling from loose to tight and reduces the application system's degree of distribution.

The initial schemata represent a business application system with a maximum degree of distribution related to the decomposition of a given business process model. Applying this design step repeatedly leads to a desirable degree of distribution with respect to redundancy and consistency.

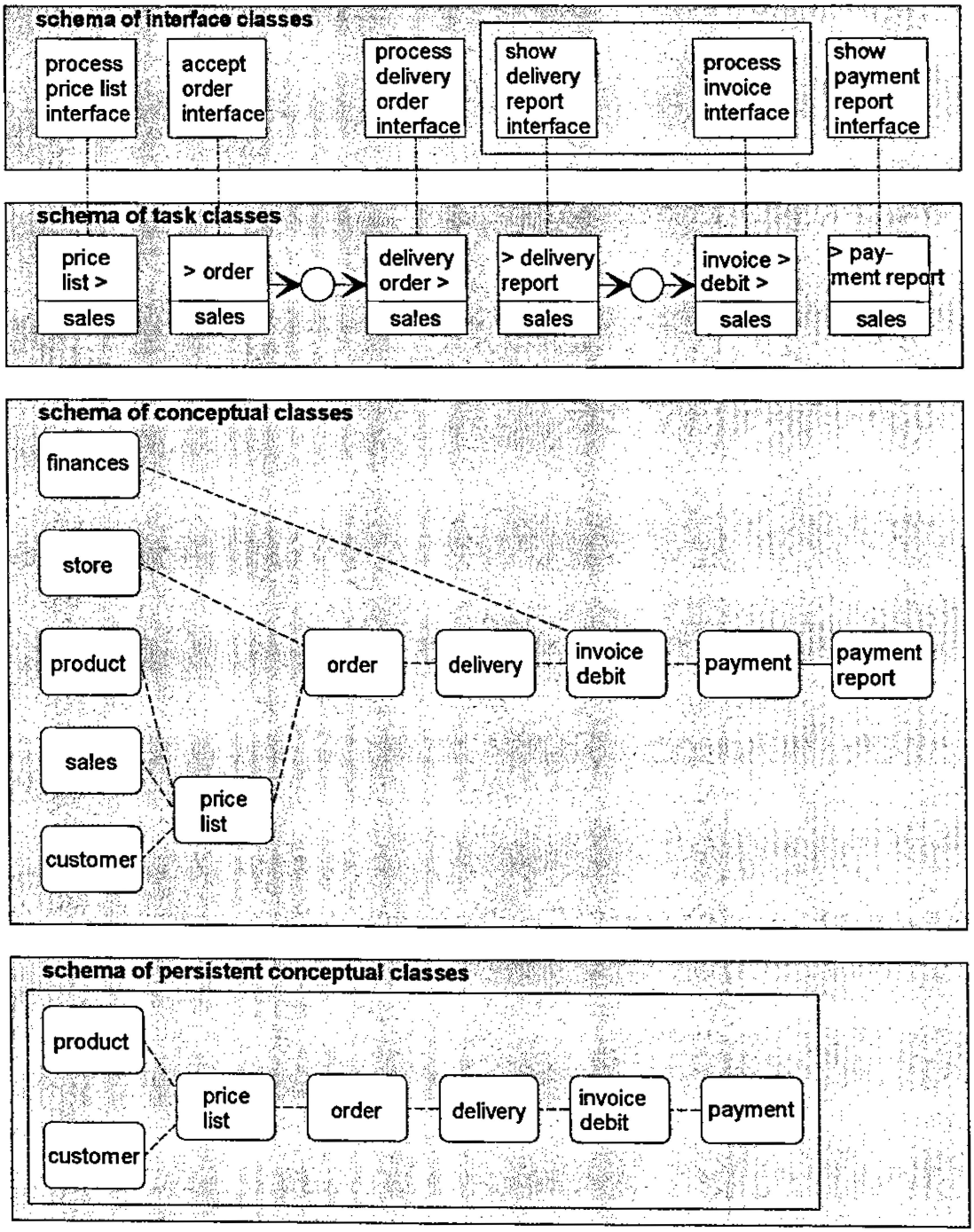


Fig. 12: Components of the domain-related architecture

Figure 12 shows the resulting schemata after applying the design rules:

- Assigning attributes and operations to the conceptual classes in figure 11 uncovers that some classes show (nearly) the same attributes and operations. To avoid redundancy and inconsistency, these classes should be combined. Refering to figures 11 and 12, the classes *order* and *delivery order* are combined to *order*; *delivery* and *delivery report* are combined to *delivery*; and *debit* and *invoice* are combined to *invoice/debit.*

- The structure of the initial schema of task classes is shown above in figure 8. It contains a task class for each task of the business object *sales*. To avoid redundancy and inconsistency again, the tasks *invoice>* and *debit>* have been combined (see figure 11). This is because the tasks perform nearly the same function (redundancy), and to ensure the *invoice* and *debit* are sent to *customer* and *finances* at the same time (consistency).

- The initial schema of interface classes provides an interface class for each task class (see fig. 12). Here, the *show delivery report* interface and the *process invoice* interface are combined. This avoids redundancy when processing invoices on delivered products by a person using the application system.

- Assuming that the conceptual classes *finances*, *store*, and *sales* have no persistent attributes, and that *payment report* can be derived from *payment*, leads to the initial schema of persistent conceptual classes shown in figure 12. If it is required to manage the referential integrity constraints between the classes by a database management system (consistency), the classes have to be combined into a single database schema.

4.3. Specification of the Physical Architecture of a Distributed Business Application System

The third layer of figures 2 and 9 specifies the physical architecture of a distributed business application system. At this layer, the domain-related components of the second layer are assigned to virtual machines, which in turn are assigned to lower level virtual machines.

Virtual machines are actors for the domain-related components of the second layer. Examples of virtual machines are database management systems which support persistent conceptual classes, user-interface management systems which support interface classes, and other application-independent class libraries.

The design goal of this step focuses on the criteria *capacity and performance of virtual machines*. This goal is achieved by a convenient mapping of domain-related components into virtual machines. In this context it may be necessary to reduce the number of components and

relationships at the domain level. This is done by a further combination of domain-related classes. In contrast to the design step at the second layer, classes of different schemata may also be combined.

The design step at the third layer is highly dependent on the specific hardware and software platform and therefore cannot be described in general.

5. Conclusion

This paper focuses on distributed business systems. It considers a distributed business system to be a set of business processes, coordinated by negotiation and feedback control principles. The decomposition of a business system is directed by the outcome of the business processes and follows the client/server concept.

Business application systems are regarded as actors for business processes. The linkage of business processes to business application systems ensures compatible structures at both layers. An initial structure of a business application system can even be derived from the corresponding business process model. The structuring of a business application system follows the paradigms of (1) the coordination principles in business processes, (2) the application/data management/communication structure, and (3) the client/server concept. It is noteworth that the object-oriented paradigm and the client/server concept is valid over all three layers of the SOM approach.

The specification of a distributed business application system remains challenging. We hope that the approach outlined in this paper contributes to turn the specification of a distributed business application system from an intuitive trial and error process to a systematic system engineering procedure. This is the most important prerequisite for total quality management at a high level of maturity.

References

Enslow, P.H.: What is a 'Distributed' Data Processing System? In: IEEE Computer, Vol. 11, No. 1, January 1978, 13 - 21

Ferstl, O.K.; Sinz, E.J.: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik Band 32, Heft 6 (1990), 566 - 581

Ferstl, O.K.; Sinz, E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik Band 33, Heft 6 (1991), 477 - 491

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

Ferstl, O.K.; Sinz, E.J.: Geschäftsprozeßmodellierung; In Wirtschaftsinformatik Band 35, Heft 6 (1993), 589-592

Ferstl, O.K.; Sinz, E.J.: Grundlagen der Wirtschaftsinformatik. 2. Auflage, Oldenbourg, München 1994

Ferstl, O.K.; Sinz, E.J.; Amberg, M.; Hagemann, U.; Malischewski, C.: Tool-Based Business Process Modeling Using the SOM Approach. Proc. IFIP World Computer Congress, Hamburg 1994

Gray, J.; Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, California 1993

Martin, B.E.; Pedersen, C.H.; Bedford-Roberts, J.: An Object-Based Taxonomy for Distributed Computing Systems. In: IEEE Computer, Vol. 24, No. 8 (1991), 17 - 27

Acknowledgement:

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