IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4481

Identifying Business Components Using Conceptual Models

Klement J. Fellner and Klaus Turowski
Otto-von-Guericke-University Magdeburg, PO box 41 20, 39016 Magdeburg,Germany
Tel: 49 391 671-83 86; Fax: 49 391 671-1216; {fellner | turowski}@iti.cs.uni-magdeburg.de

ABSTRACT

In general, a business component is a software unit that carries out business tasks in a given business area, e.g. production planning, accounting, or materials management. Using business components may lead to an economic make and buy of business application systems by combining advantages of developing individual software with those of standardized off-the-shelf software. However, the necessary standardization of open and flexible business components to obtain these advantages is still missing. On the other hand, the information systems community has done a lot of research to acquire conceptual models that describe certain business application areas. In this paper, we propose an approach to utilize this work in order to identify business components.

1 BUSINESS COMPONENTS

Increasing productivity of system development and augmenting flexibility of software systems to react to business process changes has become a major concern for companies. A business application system supporting these processes therefore benefits from being built from well-defined, reusable (off-the-shelf) components. This allows application designers to avoid drawbacks usually emerging in individual software development (e.g. high potential of failure, unpredictable costs) as well as pitfalls traditionally associated with standardized software for *enterprise resource planning* (ERP-software) (e.g. functional ballast, high training costs). Additionally, a plug and play scenario is applicable that uses off-the-shelf-components mixed with individually implemented components known as *make and buy* (Kurbel et al. 1994).

A software component, in general, is defined as a software unit, whose behavior is clearly defined through a publicly exposed interface, whose implementation is hidden, and that is deployable as a self-contained unit. For an in depth discussion about the differences between (business) objects and software components in general cf. (Szyperski 1998, pp. 30-32). The term (software) *component* itself is defined differently in almost every approach, but with a similar core statement. For differences in definitions cf. e.g. (Nierstrasz 1995), (Orfali/Harkey/Edwards 1996, p. 38), (Brown/Wallnau 1996), (Sametinger 1997, p. 68), (Szyperski 1998, p. 30). (Fellner/Turowski 2000) provides a classification framework.

As a *guiding model*, black-box software components are composed using other software components (even from different suppliers) to a customer-individual application system. When applying the concept of software components to business application domains, we talk about *business components* (BC). According to (Fellner/Turowski 2000), we use the following definition of BC:

> A *business component* (BC) is a reusable, self-contained and marketable software unit providing services out of a business application domain through a well-defined interface and which can be deployed in configurations unknown at development time.

To compose an application system with BCs from different suppliers, a description of each BC in an independent specification language is necessary. These descriptions serve as *software contracts* encompassing several levels of abstraction (Beugnard et al. 1999), e.g. syntactic, behavioral, synchronization, and quality of service level. Today, we need several kinds of notations to wholly specify a BC, e.g. on the syntactic level the Interface Definition Language (IDL) as suggested by the Object Management Group (OMG) (OMG 1998, S. 3.1-3.40), on the behavioral level the Object Constraint Language (OCL) as suggested in (Rational Software et al. 1997), or temporal logics for the synchronisation level as suggested in (Saake 1993).

Besides specifying BCs, their extent regarding implemented business tasks has to be standardized at a level still to specify, to allow a truly independent deployment of BCs (Turowski 2000). In the following, we give some examples of ongoing standardization efforts concerning the business application domain:

- The Object Management Group (OMG) proposes a standard business model for selected business areas (Domain Services) based on their component model (OMG 1997). However, the OMG's component model primary focuses on technical cooperation of objects, based on the *common object request broker architecture* (CORBA), rather than on cooperation at the business level. Especially, it is not specified how independent BCs are cooperating to carry out specific business functions. The application domain models are based on domain-specific *business objects* - real-world objects existing in a given business context (OMG 1996). For example, objects like *customer*, *order*, or *invoice* are defined as business objects.
- The Open Application Group (OAG) focuses on an easy and cost-effective integration of existing business application systems based on a middleware application program interface (API) and the specification of so-called Business Object Documents (BOD) (OAG 1997). With this, integration of coarse-grained components (modules of standardized ERP-systems) is addressed.
- SAP, as one of the largest providers of integrated business application systems, defines business objects as a public interface to their SAP R/3 system. Along with the publication of their interface, SAP is restructuring the previously known SAP-modules (e.g. FI, CO, MM, PP, etc.) to components (SAP 1997b). This indicates an attempt to establish an (proprietary) industry standard, allowing third-party suppliers to make their software systems SAP-aware.

In addition, there are other efforts to develop standards for BCs based on Java, as the San Francisco project at IBM (IBM 1997), or Sun's Enterprise JavaBeans (Sun Microsystems 1999a).

A lot of work is also done in the research field of *reference models*. Reference models give recommendations for the design

of information systems at the business level, e.g. (Scheer 1994). These contributions can be classified as preliminary, but nevertheless necessary for component-oriented software development, since reference models classify and arrange tasks at the business level without defining any software components or proposing how to derive them.

2 OPEN RESEARCH TASKS

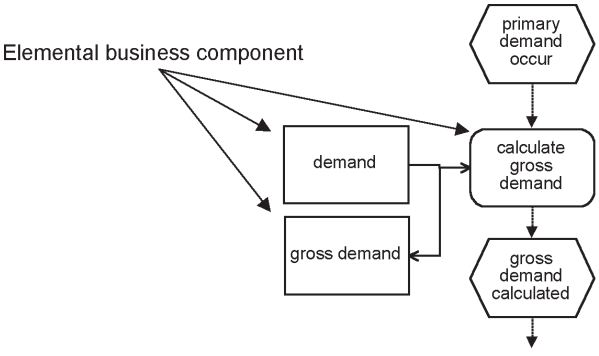
Before specifying BCs an identification and standardization process has to be carried out, which is part of the *domain analysis* (Prieto-Díaz 1987). Despite different current efforts focussing on identifying BCs for business application domains, commonly accepted results are still missing. However, a closer look to business domains reveals that a lot of research work has already been done acquiring *conceptual models* describing these domains, e.g. (Mertens/Griese 1993), (Scheer 1994), (Becker/Schütte 1996), (Mertens/Griese 1997). Representing a snapshot of certain application domains, these models can serve as a starting point to derive potential BCs. The steps from a given conceptual model to BCs will be shown in detail in the next section. We do therefore not propose yet another standardized component model, but provide a procedure which eases and supports any given standardization endeavor.

Another problem that has to be addressed when identifying BCs is the right *granularity*. BCs may be defined at every level between a coarse grained granularity, like materials management, or a fine level, like stock keeping, calculating demands, and administering bills of materials. At the lowest level, BCs are equal to the processed data objects, e.g. bill of materials, part, demand, asset account, etc.

Depending on the level at which business objects are defined, several assets and disadvantages are identifiable. Coarse-grained BCs, like materials management, limit customers as well as suppliers. Customers, who only need a certain functionality provided by a given BC (e.g. stock keeping) in addition to an existing business application system have to deploy the whole BC with all the functional overload. Suppliers of coarse-grained BCs have to design, implement, and test a relative huge BC hindering a short time to market, which is a key success factor in the ERP-software business. On the other hand, too fine-grained BCs may have negative influence on the system's performance, because a large number of fine-grained BCs, neatly connected through well-defined interfaces, is necessary to support a business process.

As a conclusion, BCs of relatively fine granularity should be standardized to allow suppliers to assemble and sell them as specific and competitive components. To make this conclusion operable it is necessary to develop a way to measure the appropriateness of the granularity from the domain, and as a trade-off, from the technical point of view.

Figure 1: *Identifying BCs*



Furthermore, the problems of task- and data-related redundancies, communication, and coordination between BCs have to be addressed. Interfaces at a technical and a business level connect the parts of a component-based application system. Technical interfaces mainly manage communication between components. Interfaces at the business level support business task-related communication at a semantic level. This allows a coordinated cooperation of different BCs to execute a given business task.

At the business level two different types of business-related dependencies, implemented as interfaces of a BC, may be distinguished: One kind leads to interfaces that are responsible for accessing data objects, the other type triggers business functions. Assuming that two BCs, *materials requirements planning* and *inventory management*, have to cooperate in order to cover the functional demands of the materials management of a given company. Both BCs are using part-related data, since a part is the basic element on which demand is calculated. Inventory management calculates the stock and the reserve stock on the part data. Moreover, materials requirements planning has to use information, e.g. future stock of a given part that have to be calculated by inventory management. For this reason, the inventory management BC has to provide a standardized functional interface that can be used by the materials requirements planning BC.

By definition, BCs offer services, but do not know anything about other BCs. Therefore, other software parts are necessary to support the execution of processes at the business level and to solve rising *conflicts*. Furthermore, a conflict solving method is necessary if deployed BCs partially overlap, e.g. two different BC provide the same function *calculate future stock*, or both contain objects of the type *part*. Suitable middleware assuring the cooperation at the technical level has to be specified. This middleware has to solve the rising conflicts by choosing the *right* functions and the *right* data. Consequently, we will focus on the identification task of BCs. Although the approach presented herein does not directly solve other mentioned problems, it will support the solution finding process (e.g. to find conflicting BCs).

3 IDENTIFYING BCS USING CONCEPTUAL MODELS

A standard for BCs should have at least the following characteristics. It should provide maximum *flexibility* for customers as well as for suppliers. For the supplier it is important that he has a high degree of freedom defining his BCs to gain an advantageous market position and to provide components for similar or even intersecting application areas as other suppliers. From the customer's point of view, BCs should be easy to exchange as a whole or in parts to support business changes. For these reasons, it has to be assured that any given set of BCs may work together and that eventually arising conflicts are solved automatically.

To assure that BCs are freely exchangeable and conflicts are solvable, the use of *elemental BCs* is suggested. By definition, a BC may be built out off other BCs. An *elemental business component* (eBC) is defined as a BC that is not further subdivided. It is atomic in the sense that there does not exist any "smaller" BC sharing a subset of data or functions that are covered by any eBC. Elemental BCs are the basic (logical) units that are used to solve conflicts and to offer data or functionality. In contrast to business

objects, an eBC may contain, implement, or produce several of these real-world objects, depending on their interdependence and granularity. However, in some case an eBC may appear to be a factory that creates and manages a certain business object. Assume, e.g., a business object *order*. This may be an instance of a class *order*. In contrast, an eBC *order* would offer services comparable to a database management system, which is specialized to create, hold, and search entities of the type *order* (which in reverse might be business objects of the type *order*).

In the following, eBCs are primarily used to standardize interfaces and services and to allow the building of supplier-specific BCs by assembling BCs out of eBC. The suggested approach is based on two hypotheses:

- Elemental BCs can be derived out of existing conceptual models.
- Cooperation and conflict solving between any market-relevant BCs may be achieved using eBCs.

This contribution focuses on the first hypothesis. The second hypothesis is discussed in detail in (Fellner/Rautenstrauch/Turowski 1999).

The following example, which describes the proposed approach, uses the well-known reference model given in (Scheer 1994) as starting point for the identification of eBCs. Elemental BCs are identified and generated out of *extended Event driven Process Chain* (eEPC) diagrams (Keller/Nüttgens/Scheer 1992, pp 32-35) and the corresponding data models, which are part of the reference model. We use a simplified representation without organizational units and entity attributes to illustrate the procedure on the basis of an extract of a materials requirements planning (MRP) model (Scheer 1994, p. 170) (fig. 2a):

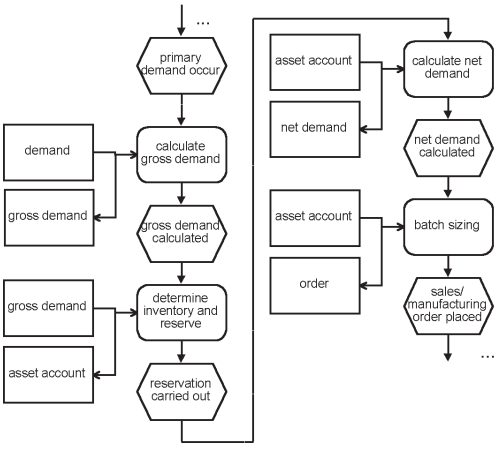
- Each function object and information (entity) object in the eEPC is mapped to a corresponding BC (fig. 1).
- As result, a *preliminary* set of eBCs is generated (fig. 2b). In contrast to object-oriented concepts, applying classification is used to identify eBCs. The model parts of the eEPC are divided into functional and information objects to simplify the identification task (cf. Turowski 1997, pp. 100). Functional objects represent dynamical aspects or business tasks, whereas information objects describe application or control data. In contrast to common object-oriented approaches, data and functions are not somehow recombined in later steps. The idea to distinguish between different kinds of objects can be found as well in literature, e.g. Jacobson distinguishes between interface, entity, and control objects (Jacobson 1995, pp. 174-200), Müller-Luschnat et al. suggest task and storage classes (Müller-Luschnat/Hesse/Heydenreich 1993, pp. 78-86), and Kueng et al. use role, business, and storage classes (Kueng/Bichler/Schrefl 1996, p. 48).
- To identify further eBCs, a more detailed data model, which corresponds to the process model used so far and that is part of the reference model, is taken into account. The data model is presented in a separate view within the reference model. It contains the information objects given in the eEPC. In addition, it encompasses the relation between the respective information objects as relationship types. Information objects given in the eEPC normally correspond to entity types in the data model. The data model is used to examine eBCs in more detail, which are related to information objects. The business tasks are not further subdivided. Each of the identified business tasks becomes an eBC. However, this does only work, if a specific *abstraction level* of the reference model is used. In common, reference models encompass different views (e.g. data view or process view) and different abstraction levels. Choosing the right abstraction level is crucial for the proposed approach, as this effects the identification of eBCs. In general, the following abstraction levels are distinguished: With respect to the data view, one distinguishes between cluster, entity, and attribute level. With respect to the process view, one distinguishes between business process, business function, and task level. For the proposed approach, the second level of abstraction (entity/business function level, cf. fig. 3 or fig. 4) works best, since eBCs become to fine-grained on attribute level and to coarse-grained on cluster level.
- First, generalization and specialization relations are examined in the data model. Doing so, reveals that *gross demand* as well as *net demand* are both special kinds of *demand*, leaving the single eBC *demand* that replaces the eBCs *gross demand* and *net demand* (cf. fig. 3a). This is done to achieve the smallest reasonable set of eBCs.
- Next, relationships are examined in the data model. Relationships lead to eBCs, too. Relationship types are considered to be eBCs, because they often become interpreted as entities, e.g. order may also be seen as relation between customer, parts, and time, or have attributes of its own. Thus, they are potential candidates for being replaced by other BCs from different vendors. With this, the preliminary set of eBCs has to be extended by *coverage*, *demand coverage*, and *demand derivation* (cf. fig. 3a).
- Last, the *final* set of eBCs is created (cf. fig 3b).

After all (relevant) eBCs are identified, they are composed in a certain way to form a BC that supports a (sub-)set of a given business process. If a certain eBC should be added to a BC depends on the underlying application domain and has to be taken into consideration for every eBC. At this stage, any identified eBC may be used in any BC where it suits. Therefore, when these "first-cut" BCs are built up, they have to be thoroughly examined for contained eBCs. Fig. 4a depicts a possible allocation of eBCs in higher level BCs. It also shows that some eBCs may be implemented in more than one BC depending on the business area a model is built for.

To discover redundant eBCs multiple occurrences of the same eBC are eliminated

Figure 2: *Part of an eEPC diagram for materials requirements planning*

(a)

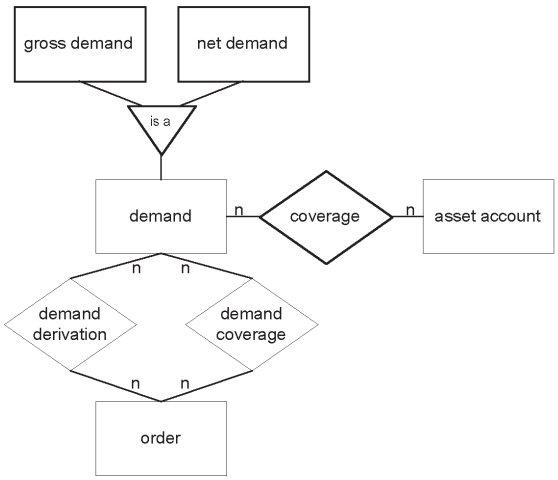


(b)

Elemental business components
Calculate gross demand
Determine inventory and reserve
Calculate net demand
Batch sizing
Demand
Gross demand
Asset account
Net demand
Order

Figure 3: *Data model and derivation of the final set of eBCs*

(a)



(b)

Elemental business components
Calculate gross demand
Determine inventory and reserve
Calculate net demand
Batch sizing
Demand
Demand derivation
Demand coverage
Asset account
Coverage
Order

(cf. fig. 4b). The result is the graphical representation of fig. 3b with the additional information of overlapping BCs. Only for those eBCs appearing in more than one BCs (*asset account* in the example) conflicts can occur and have to be solved by defining a clear strategy which BC manages the concerned eBCs. Elemental BCs appear in more than one BC if they are shown in an overlapping region. Any other BC can access this eBCs through the interface provided and implemented by the managing BC. Hence, it follows that using eBCs allows an unequivocal identification of overlapping BCs as well as fast conflict detection and solving.

4 EVALUATION

We implemented an application system for production planning and control, which uses the described approach to identify eBCs, as a research prototype. BCs are composed of previously identified and implemented eBCs. Their identification was carried out following the mentioned steps. Besides the eBCs mentioned above, additional eBCs were implemented, e.g. *customer order* or *bill of materials*. The actual procedure was also chosen to test the interchangeability of specific eBCs by providing parts of the functionality through legacy systems. In our prototype implementation, we use SAP R/3 (Rel. 4.0b) to show and test integration aspects.

Implemented in Java and composed using IBM's Visual Age for Java, the prototype was produced using state-of-the-art technology and systems. One of the major reasons why Java was chosen, is the availability of a component API, the JavaBeans (Sun Microsystems 1997), which are by definition self-contained components.

To show the integration of existing systems, we used the Business Application Programming Interface (BAPI) (SAP 1997a) of SAP to connect to their system R/3. Here we found a way to reuse existing components by means of the InfoBus technology from Lotus (Sun Microsystems 1999b) as well as Java components (Enterprise Access Builder) from IBM, which allow a fast and easy first-cut connection to R/3.

First tests brought promising results. The implemented eBCs are interchangeable, as long as they provide the specified services. However, the composition of a BC consisting of a huge number of eBCs may become confusing. Therefore, we work on an approach to compose eBCs providing more (and conflicting) services. To allow eBCs with conflicting services, a mechanism has to be provided to solve content-related conflicts. Therefore we proposed an approach using a special kind of mediating components in (Fellner/Rautenstrauch/Turowski 1999).

The problem of finding BCs is closely connected to the problem of finding the *right* object, which is well-known from the area of object orientation (cf. e.g. Kurbel/Teubner 1996, pp. 243-245). An approach that is related to our contribution may be found in (Bungert/Heß 1995). There, the authors suggest an approach to identify (object) classes from conceptual models. Another approach is given in (Ferstl et al. 1997, pp. 38-44), which analyses process models based on Petri Nets.

5 OUTLOOK AND CONCLUSIONS

Using eBCs offers a possible way to build flexible supplier specific BCs for the area of business application systems. They allow suppliers to offer BCs that cover partially intersecting application areas, provide an easy way to exchange BCs or parts of BCs, help to solve conflicts, and can be identified using *existing* conceptual models. An approach to identify eBCs using reference models was discussed in this paper together with an interim definition of eBCs from the area of materials management, which has to be further expanded to a model covering the whole area of production planing and control.

Besides further extension and evaluation of the existing prototype, future work will also include using standardized specification languages as discussed in the first section. So far, interfaces are solely defined using Java. In addition, standardization at the attribute level as well as standardization of interfaces and services will be taken into account.

REFERENCES

Becker, J.; Schütte, R. (1996): Handelsinform-ationssysteme. Landsberg.

Beugnard, A.; Jézéquel, J.-M.; Plouzeau, N.; Watkins, D. (1999): Making Components Contract Aware. IEEE Computer 32(7), pp. 38-44.

Brown, A. W.; Wallnau, K. C. (1996): Engineering of Component-Based Systems. In: Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. Ed. A. W. Brown. Los Alamitos, California, pp. 7-15.

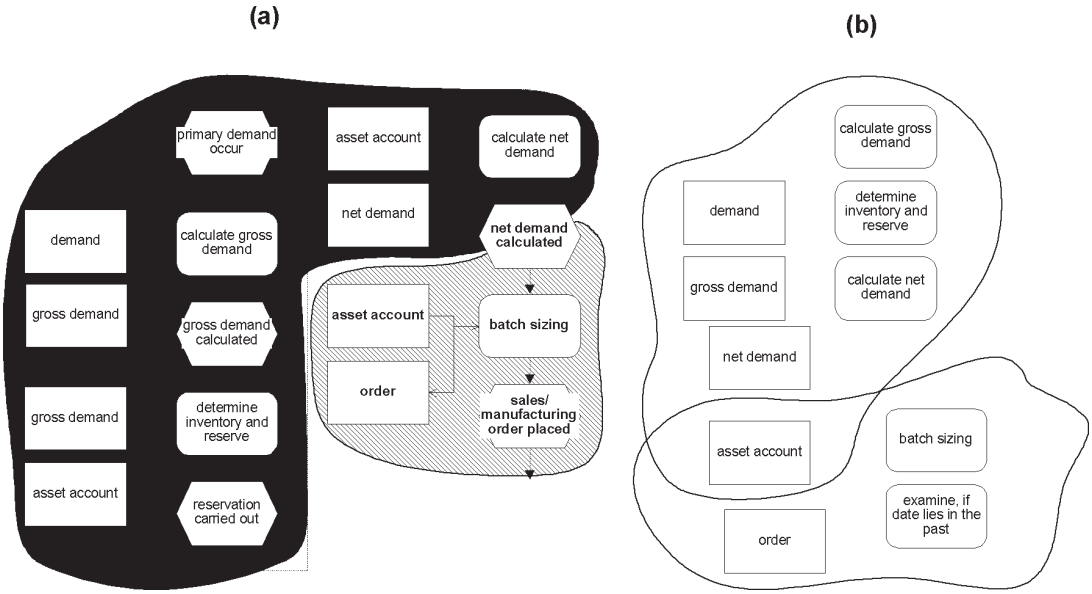
Bungert, W.; Heß, H. (1995): Objektorientierte Geschäftsprozeßmodellierung. Information Management 10(1), pp. 52-63.

Fellner, K.; Rautenstrauch, C.; Turowski, K. (1999): Fachkomponenten zur Gestaltung betrieblicher Anwendungssysteme. IM Information Management & Consulting 14(2), pp. 25-34.

Fellner, K.; Turowski, K. (2000): Classification Framework for Business Components. To appear in: R. H. Sprague (Ed.) Proceedings of the 33rd Annual Hawaii International Conference On System Sciences. Maui, Hawaii, (CD-ROM), 10 pages.

Ferstl, O. K.; Sinz, E. J.; Hammel, C.; Schlitt, M.; Wolf, S. (1997): Bausteine für komponentenbasierte Anwendungssysteme. HMD 34(197), pp. 24-46.

Figure 4: *Possible allocation of eBCs in higher level BCs and overlapping BCs*



IBM (Ed.) (1997): San Francisco Project Technical Summary. http://www.ibm.com/Java/Sanfrancisco/prd_ summary.html. Accessed: 16.06.1998.

Jacobson, I. (1995): Object-Oriented Software Engineering. 6. ed., Wokingham.

Keller, G.; Nüttgens, M.; Scheer, A.-W. (1992): Planungsinseln - Vom Konzept zum integrierten Informationsmodell. HMD 29(168), pp. 25-39.

Kueng, P.; Bichler, P.; Schrefl, M. (1996): Geschäftsprozeßmodellierung: Ein zielbasierter Ansatz. Information Management 11(2), pp. 40-50.

Kurbel, K.; Rautenstrauch, C.; Opitz, B.; Scheuch, R. (1994): From »Make or Buy« to »Make and Buy«: Tailoring Information Systems Through Integration Engineering. Journal of Database Management 5(3), pp. 18-30.

Kurbel, K.; Teubner, A. (1996): Integrating Information-system Development into Business Process Reenginering. In: C. Nagib; C. Callaos (Eds.): Proceedings of the International Conference on Information Systems Analysis and Synthesis. Orlando, pp. 240-246.

Mertens, P.; Griese, J. (1993): Planungs- und Kontrollsysteme in der Industrie. Vol. 2, 7. ed., Wiesbaden.

Mertens, P.; Griese, J. (1997): Administrations- und Dispositionssysteme in der Industrie. Vol. 1, 11. ed., Wiesbaden.

Müller-Luschnat, G.; Hesse, W.; Heydenreich, N. (1993): Objektorientierte Analyse und Geschäftsvorfallsmodellierung. In: Objektorientierte Methoden für Informationssysteme. Eds.: H. Mayr; R. Wagner. Berlin, pp. 74-90.

Nierstrasz, O. (1995): Research Topics in Software Composition. In: A. Napoli (Ed.) Proceedings, Langages et Modèles à Objects. Nancy, pp. 193-204.

OAG (Ed.) (1997): White Paper: Open Applications Integration: Projects of the Open Applications Group.

OMG (Ed.) (1996): Common Facilities RFP-4: Common Business Objects and Business Object Facility.

OMG (Ed.) (1997): CORBA Component Model RFP. ftp://ftp.omg.org/pub/docs/orbos/97-06-12.pdf. Accessed: 21.01.1998.

OMG (Ed.) (1998): The Common Object Request Broker: Architecture and Specification (Revision 2.2).

Orfali, R.; Harkey, D.; Edwards, J. (1996): The Essential Distributed Objects Survival Guide. New York.

Prieto-Díaz, R. (1987): Domain Analysis for Reusability. COMPSAC 87. Tokyo, Japan, pp. 23-29.

Rational Software; Microsoft; Hewlett-Packard; Oracle; Sterling Software; MCI Systemhouse; Unisys; ICON Computing; IntelliCorp; i-Logix; IBM; ObjecTime; Platinum Technology; Ptech; Taskon; Reich Technologies; Softeam (1997): Object Constraint Language Specification: Version 1.1, 1 September 1997. http://www.rational.com/uml. Accessed: 17.04.1999.

Saake, G. (1993): Objektorientierte Spezifikation von Informationssystemen. Stuttgart.

Sametinger, J. (1997): Software Engineering with reusable components. Berlin.

SAP (Ed.) (1997a): BAPI-Programmierleitfaden. Walldorf.

SAP (Ed.) (1997b): BAPIs - Einführung und Überblick. Walldorf.

Scheer, A.-W. (1994): Business Process Engineering: Reference Models for Industrial Enterprises. 2. ed., Berlin.

Sun Microsystems (Ed.) (1997): JavaBeans: JavaBeans API Specification 1.01. Mountain View.

Sun Microsystems (Ed.) (1999a): Enterprise JavaBeans Specification, v1.1. Mountain View.

Sun Microsystems (Ed.) (1999b): InfoBus 1.2 Specification. Mountain View.

Szyperski, C. (1998): Component Software: Beyond Object-Oriented Programming. 2. ed., Harlow.

Turowski, K. (1997): Flexible Verteilung von PPS-Systemen - Methodik Planungsobjekt-basierter Softwareentwicklung. Wiesbaden.

Turowski, K. (2000): Establishing Standards for Business Components. To appear in: IT Standards and Standardisation: A Global Perspective. Ed. K. Jakobs. Hershey, pp. 131-151.