

# Component Market Specification Demand and Standardized Specification of Business Components

Alexander Keiblinger, Klaus Turowski, Johannes Maria Zaha

Chair of Business Information Systems (Wirtschaftsinformatik II)  
University of Augsburg  
Universitätsstraße 16, 86135 Augsburg, Germany  
{alexander.keiblinger|klaus.turowski|johannes.maria.zaha}  
@wiwi.uni-augsburg.de

**Abstract.** Compositional reuse of components requires standardized techniques to specify software components, especially if applications are created by combining third party components, which are traded on component markets. As in established engineering disciplines like mechanical engineering or electrical engineering, formal specification of business components that becomes part of contractual agreements is needed. Based on the examination of software markets and potential black-box component software markets the need for standards towards the formal specification of software components will be derived. The results of our working group regarding specification of business components and the theoretical foundation of component markets will be presented.

## Importance of Component Software

The idea of software systems made up from pre-fabricated software components that could be exchanged via software component markets has been on the agenda of software engineering at least since McIlroy has outlined his vision in 1968 [1]. The underlying idea is to combine components from different vendors to an application which is individual to each customer and where the compositional plug-and-play-like reuse of black box components enables software component markets. Ideally, the advantages of both standard and individual software production are combined. The principle of modular design that is underlying component based software systems is equally important for the discussion of the technological as well as the economic advantages of component based software systems. A rich literature on the general advantage of such systems exists, c.f. [2], [3], [4], [5], [6]. Modular systems have been described as the result of a functional decomposition [7] and the conception of modular systems has been thoroughly analysed by system theory.

Combining off-the-shelf software components offered by different vendors to customer-individual business application systems is a goal that is followed-up for a long time. By achieving this goal, advantages of individually programmed software

could come together with those of standardized off-the-shelf software. In this context, we are speaking about *compositional reuse* techniques. Compositional reuse is a special kind of reuse technique as *generative* techniques or *code and design scavenging* [8, pp. 25-28]. The emphasis on compositional reuse stems from our *guiding model* which is the compositional plug-and-play-like reuse of black box components that are traded on a component market. In general, a guiding model is an ideal future state that might not completely be reached. In the future a business component will be filtered out by parameterized searches using specification fragments that define properties of the needed software component. If a suitable component has been found it will be bought on a component market and integrated into the business application system. In [9, pp. 11-14] the steps are explained in detail, e.g. technical and semantic adaptation or composition. Expected improvements, which should come along with using software components, concern cost efficiency, quality, productivity, market penetration, market share, performance, interoperability, reliability, or software complexity, cf. e.g. [10 S. 29-32].

The reminder of the chapter remains as follows. After providing background information about component software and software markets, we discuss the necessity of a multi-level notation standard and present our work in this area. An example will show how the specification of business components works for the domain of personnel administration. The need for standardized specification will be summarized and the component community will be called upon for participation in the process of further refining the standardisation of business components.

## Component Markets

Business Components are software components that are designed to support specific business applications eg. Finance or CRM. The idea is to allow companies to follow up a Mix-and-Match-Strategy to build a company specific business application system from scratch by combining software components from different vendors. Business Components that are sold on electronic markets would therefore create component markets. A pre-condition is that there are accepted standards. In our work we looked at markets for software components in general and tried to find out what and why we have to standardize.

Using Web Services as an object of investigation, one can see that with all the different specification languages for Web Services, users are mainly able to specify interfaces. The specifications result mainly in a programmers view to a function call. Business or task related aspects, or even non functional aspects are mostly omitted. Not a good starting point for automated ad-hoc service composition. It lacks an integrated specification of technical aspects of a business application system as well as specification of business task related aspects. In order to efficiently trade software components via electronic markets it is fundamental to precisely describe the individual product.

We define a *component market* as an abstract place where components are exchanged

and their value is determined by the bidding process between the supplier and the user of components[11]. There needs to be maturity in the market to establish an effective market which will be accepted by component customers. A standardized specification in itself would be a kind of public good and there is need for quality standards as well as specification standards, since markets work best if buyers and sellers have the same perception of what is traded on the markets. As one can see observing today's software markets a lack of public standards leads to idiosyncratic investments and lock-in situations. If there is little incentive or pressure to agree on open standards and prevent proprietary extensions, dependence on proprietary supplier results and prevents the evolution of mature software and component markets. With respect to software industry, what are needed for mature markets to evolve are real integrative specification standards and quality standards for software components.

Specification standards and the capability of software buyers to specify their needs are crucial factors for software markets in general and software component markets. Most today's specification standards aim to support the technical perspective of software components. We call this the "specification gap". That means there is no common integrative specification approach that covers the technical as well as the business perspective of software and software components. The knowledge and capabilities of the buyer can influence the importance and reach of standards in the marketplace. The customers have to fulfill an active role in the process of market maturation. For component markets to become successful it is important to realize the role and influence of the buyer in the process of market maturation and provide rigid but flexible specification tools or languages. To close the specification gap a comprehensive methodology for different specification goals is needed. If the buyer side is able to specify demand in a standardized and unambiguous way and enforce the standards, this would strengthen their market position compared to the sellers. Buyers could play a more active role if they are able to specify their demand. There would be less need to *believe* that a third party is able to specify their demand, as it is usually the case in today's individual software projects or using off the shelf business applications.

The next section will describe the results of our working group that are heading towards such an integrative specification approach for software components. Build upon the idea of software contracts the approach uses a broader spectrum of contracts about characteristics of software to support the evolution of component software markets.

## Business Components and Software Contracts

According to [12, pp. 3-4], we define the term *component* as follows: A component consists of different (software-) artefacts. It is reusable, self-contained and marketable, provides services through well-defined interfaces, hides its implementation and can be deployed in configurations unknown at the time of development. A *business component* is a component that implements a certain set of services out of a given business domain. In order to be operable, components need a basic infrastructure, e.g. Enterprise Java Beans (EJB) or .NET, which we will call

*component system framework*. For a discussion of other component definitions or component models, refer to [13, pp. 164-168] and [12]. To use business components according to this guiding definition, it is necessary to standardize them. The interface and behaviour of a component has to be described in a consistent and unequivocal way. Specification becomes more and more important with respect to third party composition of business components, since the specification might be the only available support for a composer who combines business components from different vendors to an application system.

*Software contracts* offer a good solution to meet the special requirements of specifying business components. Software contracts go back to MEYER, who introduced contracts as a concept in the *Eiffel* programming language. He called it *programming by contract* [14] and extended it later to the concept of *design by contract* [15]. Furthermore, similar concepts are described in [16] or [17].

Software contracts are obligations to which a service donator (e.g. a business component) and a service client agree. There, the service donator guarantees that

- a service it offers, e.g. calculate balance or determine demand,
- under certain conditions, which have to be met by the service client, e.g. the provision of data necessary to process the service,
- is performed in a guaranteed quality, e.g. with a predetermined storage demand or with an agreed response time, and
- that the service has certain external characteristics, e.g. the specified interface.

[18, pp. 38-40] describes a general model for software contracts for components with four tiers. One has to distinguish between syntactic, behavioural, synchronization, and quality-of-service level. Business components need to be specified on each of these levels.

A *methodical standard* for the specification of business components is needed and would enable a common understanding of component specifications is needed. Such a standard can be achieved by identifying the objects to be specified and by choosing a mix of notations that is standardized, accepted and agreed by all participating parties. This ensures the reusability of components and the exchange of components between companies and software developers can be simplified. The *specification* of a business component is defined as a complete, unequivocal and precise description of its external view. It describes which services a business component provides under which conditions.

To fully implement the complexity of business software components, specification on different contract levels (see eg. Fig. 1) is needed. Besides arranging the specifications' contents according to contract levels, a specific notation language is needed for each level of abstraction. In the context of systematic specification of business components it is helpful to use a well-known and well-accepted formal notation, which can be used on more than one contract level. A notation is called *formal* in this context if syntax and semantics of the notation are unequivocal and consistent. For this reason, formal notations seem to be particularly suitable for the specification of software contracts.

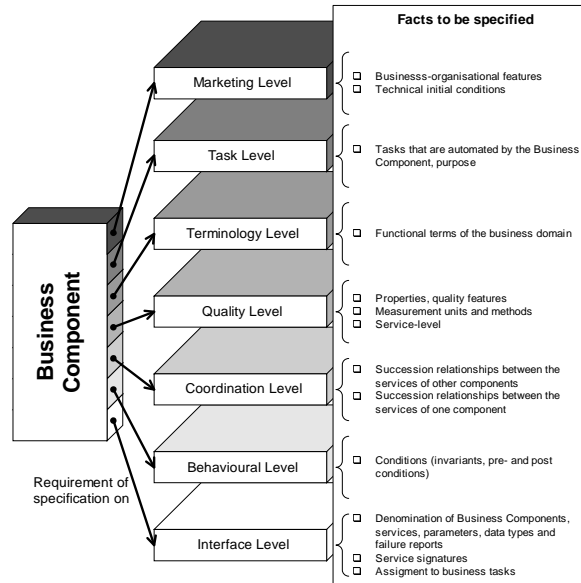


Fig. 1. Software contract levels and facts to be specified

At interface level basic agreements are concluded. Typical parts of these agreements concern names of services (offered by a business component), names of public accessible attributes, variables, or constant values, specialized data types (in common based upon standardized data types), signatures of services, as well as the declaration of error messages or exception signals. To do so, we use e.g. programming languages or *Interface Definition Languages* (IDL) like the IDL that was proposed by the Object Management Group (OMG) [19 S. 3.1-3.74]. The resulting agreement guarantees that service client and service donator can communicate with each other. However, emphasis is put on enabling communication technically. Semantic aspects remain unconsidered.

Agreements at behavioural level serve as a closer description of a business component's behaviour. They enhance the basic agreements of the interface level, which mainly describe the syntax of an interface. Agreements at interface level do not describe how a given business component acts in general or in borderline cases.

As an example, we could define an invariant condition for a business component *personal administration* at behavioural level, which says that the weekly working hours of an employee may not exceed some maximum value because of union regulations. Known approaches to specify behaviour are based on approaches to *algebraic specification* of abstract data types, cf. e.g. [20]. To describe behaviour, the specification of an abstract data type is extended by conditions. These conditions describe the abstract data type's behaviour in general (as *invariant conditions*) or at specific times (*pre conditions* or *post conditions*). In general, conditions are formulated as equations, and as axioms they become part of the specification of an abstract data type [20]. The *Object Constraint Language* (OCL) [21, pp. 6.1-6.50] is

an example for a widespread notation to specify facts at the behavioural level. It complements the *Unified Modelling Language* (UML) [21].

The coordination of functional units needs to be specified within the component itself, called *intra-component* coordination. And coordination also needs to be specified between different business components. This is called *inter-component* coordination. Agreements at *intra-component* coordination (synchronization) level regulate the sequence in which services of a specific business component may be invoked, and synchronization demand between its services. Here, e.g., in a wage payment it may be specified that the wage has to be calculated by using the hours worked before it is transferred. At *inter-component* coordination level we come to agreements that regulate the sequence in which services of *different* business components may be invoked. Here, e.g., we may define that a service *CalculateIncentiveWage*, which belongs to a business component *Personell Administration*, and which refers to a certain worker, may only be processed after a service, which belongs to a business component *Building Management* that stores the exact times of arrival and leave of each person has been questioned about the working hours of the worker.

There exist various approaches to specify business components at the coordination level. These approaches base, e.g., on using *process algebras*, *process calculi* (cf. e.g. [22]), or on using *temporal logics* (cf. e.g. [23, pp. 79-131]). In addition, (semi formal) graphical notations are in use. These are mostly graphical notations used in the context of business process modelling. Besides extended event-driven process chains (eEPC) [24, pp. 32-35] and approaches that use eEPC as a basis, e.g. [25], Petri net based notations are in use, e.g. [26]. In particular, object-oriented software development methods like [27], [28], or [29] provide such modelling means.

As an extension to functional characteristics, we have to describe *non-functional* characteristics of business components. Non-functional characteristics are specified at the quality-of-service level. Examples for these characteristics are the distribution of the response time of a service or its availability. For further non-functional requirements and their definition cf. e.g. [30, pp. 73-158].

In all levels mentioned so far, the specification of business components uses *technical terms*, which have a domain specific functional meaning (semantic), e.g. *person*, *working hours*, or *wage*. Often, these terms do not have an unequivocal meaning or definition and, hence, have to be specified to guarantee their unequivocal use. The terminology level serves as central registry for all these terms and keeps all terms that are useful for the specification and their definitions in a dictionary. In order to achieve a high quality of specification, we use norm language reconstruction to specify the respective issues [31]. We use the same technique to specify issues at the task level. There, we explain which *business tasks* are supported or automatically done through services offered by a business component.

At the marketing level we finally specify features of the business component that are important from a *business-organizational* point of view using tables, e.g., (legal) contract terms, version, coarse business domain, or vendor contact persons.

The next section will demonstrate the practical use of formal specification at the example of a business component for personnel administration. The standardized specification of business components is exemplified according to the memorandum

“Standardized Specification of Business Components” by the working group 5.10.3 “Component Oriented Business Application Systems” of the German Society for Informatics (Gesellschaft für Informatik - GI). This memorandum is a joint effort of our working group and the German business component community.

## Specification of Business Components

In the following, we use the example business domain of personnel administration to show how to specify business components according to the memorandum “Standardized Specification of Business Components”[32].

For this matter we survey the description levels downwardly from Marketing Level to Interface Level. Figure 2 shows the specification of a component *Personnel Administration* on Marketing Level, which includes the characteristics of the component from a business-organizational point of view.

<b>Name</b>	Personnel Administration
<b>Identification</b>	UUID 8DE2DFA9-3344-4340-B21D-463E61A6AE2F
<b>Version</b>	1.0
<b>Bench of Economic Activity</b>	Independent of domains
<b>Domain</b>	Human resource
<b>Scope of Supply</b>	hr_expert.jar: File of java classes for the implementation createdb.sql: Script to generate a Postgresql data base postgresql-7.0-nt-binaries.tar.gz: source code of Postgresql data base apache_2.0.45-win32-x86.msi:source code of Apache Webserver
<b>Component Technology</b>	Java 2 Platform, Enterprise Edition (J2EE)
<b>Systems Requirement</b>	Processor architecture: x86 RAM: 1024 MB Hard disc: 100MB Operating system: Windows 2000 Advanced Server Data base system: Postgresql 7.0

**Figure 2:** Specification on Marketing Level

The attributes *Identification* (an unambiguous label to identify the component) and *System Requirements* are two of various optional attributes on the Marketing Level. The remainder of the attributes exemplified are obligatory. *Name* is an alphanumerical marketing name, *Version* gives the version as well as the release of a component and *Branch of Economic Activity* the range of application of the component from a economic point of view, based on the International Standard Industrial Classification of All Economic Activities [33]. *Domain* allows a rough content-related classification and besides *Branch of Economic Activity* it is the only

attribute were several characteristics can be chosen. *Scope of Supply* names the delivered (software) artefacts. *Component Technology* names the used component technology and the underlying version standards for this component.

Every business component is used to support or execute different business tasks. The purpose of the Task Level shown in figure 3 is to associate the domain named on the Marketing Level with the business tasks and on the other hand to establish a connection between the business task (and its functional decomposition in subtasks) and the offered services on Interface Level. To get a revisable and unambiguous specification, the construct of reconstructed functional languages [31] is used. This notation recommends patterns like “has a” or “consists of” which are used to express the business tasks and subtasks.

An EMPLOYEE has an ACCOUNT
An EMPLOYEE is part of a DEPARTMENT
A BUSINESS COMPANY consists of DEPARTMENTS
An EMPLOYEE does receive a WAGE
...

**Figure 3:** Specification on Task Level

The Task Level as well as the other levels of a specification use terms, whose meaning is generally not unequivocal. The meaning of the term “employee” can be used to exemplify this problem. People working in a company can have different work contracts. Depending on the underlying definition they can be fully employed workers, part time personnel or external consultants. To provide a standardized understanding of concepts, the Terminology Level is used as the central system of concepts for a business component. In this dictionary all (important) terms used in the specification are defined (cf. figure 4).

**Figure 4:** Specification on Terminology Level

BUSINESS COMPANY	<ul style="list-style-type: none"> <li>• Short definition: BUSINESS COMPANY = DF commercial organization</li> <li>• Long definition: BUSINESS COMPANY = DF an organisation made up of people who work together for purpose of business or trade</li> </ul>
DEPARTMENT	<ul style="list-style-type: none"> <li>• Short definition: DEPARTMENT = DF unit in a BUSINESS COMPANY, which has a certain function</li> <li>• Examples: research and development, marketing, sales</li> <li>• Predicators: <math>x \in \text{DEPARTMENT} \Rightarrow x \in \text{BUSINESS COMPANY}</math></li> </ul>
EMPLOYEE	<ul style="list-style-type: none"> <li>• Short definition: EMPLOYEE = DF is a physical person, who has an official duty towards his employer</li> </ul>
WAGE	<ul style="list-style-type: none"> <li>• Short definition: WAGE = DF basic salary received by the EMPLOYEE from the employer for the supplied manpower</li> </ul>

The definition of the terms must be given in a short form and can be complemented by a long form, examples and predicators. To illustrate related terms graphically, a UML class-diagram can be used additionally.

The layers of specification presented so far focus on the specification of functional properties of business components. On the Quality Level non-functional properties are depicted. Namely suitable quality criteria, appropriate measures and methods for their actual measurement and service level agreements of the services during runtime. As an example the response time per data record could have to be less than three seconds (reference environment: cf. Marketing Level). The identification of this quality criterion could be accomplished with the Goal-Question-Metric Model (GQM) [34]. The Quality Level is in great demand of research. Currently members of our research group develop a more formal specification method for describing non-functional properties.

The last three description layers are introduced in the oppositional direction. We start with the Interface Level and go upwardly over Behavioural to Coordination Level. Figure 5 shows the specification of a component *Personnel Administration* in OMG IDL as interface definition language [19 S. 3.1-3.74].

**Figure 5:** Specification on Interface Level

```
interface PersonnelAdministration {
    typedef double Amount;
    typedef double EmployeeNo;

    struct Employee {
        EmployeeNo n;
        double AccountNo;
        double HoursWorked;
        double Wage;
        string Department;
        ...
    };

    exception TooManyHours {}

    void AddUser (in Employee e);
    void Credit (in Amount a, in Employee e);
    amount CalculateIncentiveWage (in Employee e) raises
    (TooManyHours);
};

interface extern {
    typedef short WorkingDays;

    struct Period {
        unsigned short Month;
        unsigned short Year;
    };

    WorkingDays CalculateWorkingDays (in Date d);
    ...
};
```

By using the keyword *interface*, the name of the component is determined. The Business Component depicted in figure 5 supports business tasks from the area Personnel Administration. Subsections of the Business Component can be identified unequivocally by the name of the component. *PersonnelAdministration* : : *Credit* e.g., indicates that a service *Credit* can be invoked that is part of the Business Component *PersonnelAdministration*. Subsequently, the simple (*Amount* and *EmployeeNo*) and structured types (*Employee*) are declared by using predefined types, which are necessary for the specification of interface signatures provided by this service. After defining special data types of the service provider, exceptions can be declared (key word: *exception*) that are used to report special failure situations of the service provider. In the example, a signal is declared that indicates that the entered amount of working hours is too great. The offered services of the Business Component are declared with the interface signature, which includes the name of the service, the type of the return value and the expected input parameters. Finally, the services required by the Business Component are specified (key word: *interface extern*), as well as their calling parameters and return values.

The interface level is completed with two tables which show the corresponding (functional) terms, the (data) types and the tasks and services. The corresponding levels providing the information about tasks and (functional) terms can be found on Task and Terminology Level.

(Functional) Term	(Data) Type
Wage	Wage
Employee	Employee
...	...

Task	Service
An EMPLOYEE does receive a WAGE	Credit (in Amount a, in Employee e);
An EMPLOYEE has an Account	AddUser (in Employee e);
...	...

**Figure 6:** Specification on Interface Level

In figure 7 we show a detailed description of the required behaviour of the Business Component by using the OCL, which is part of the UML and was adopted by the OMG as a standardized notation. This completes the basic specification of the Interface Level, which only describes the Syntax of the interface and leaves it open, how the component behaves in a certain situation.

**Figure 7:** Specification on Behavioural Level

```

PersonnelAdministration
    self.Employee -> forall (e:Employee | e.wage >= 0)

PersonnelAdministration : : CalculateIncentiveWage (e: Employee) :
amount

```

```

Pre:  self.Employee -> exists (f:Employee | f.EmployeeNo =
e.EmployeeNo)
Post:  result = if e.HoursWorked < 100
          then e.HoursWorked * e.Wage
          else raise TooManyHours ()
        endif

```

At first, the name the context to which the respective specification refers is named. The context is emphasized by underlining. The first condition in figure 7, e.g., refers to the business component *PersonnelAdministration* as a whole and testifies that for each *Employee* the wage must not be negative. Conditions appear either as pre conditions (keyword *pre*), as post condition (keyword *post*), or as invariant condition (no keyword). The second part of our example is delimited to the service *CalculateIncentiveWage* which is provided by the component. This reference is made by the `::` before naming the service and the expected parameters. At last, the return value of the service is mentioned, if existing. The precondition states that a dataset of the type *employee* must exist, whose value of the attribute *EmployeeNo* is equivalent to those of the delivered dataset. The post condition assigns how the return value of the service (*result* corresponds to *amount*) is computed. For this purpose the variable *HoursWorked* is checked, if its value is smaller than 100. If this is not the case, the exception *TooManyHours* is thrown. Otherwise the values of the variables *HoursWorked* and *Wage* of the dataset are multiplied and returned as result.

With the OCL it is possible to describe conditions (invariants, pre- and post conditions) for one single service of a Business Component. To constitute succession relationships between several services of one or other components, the OCL has been extended with an addition of temporal operators [35]. Figure 8 depicts an example for two services of the component *PersonnelAdministration*.

```

PersonnelAdministration :: Credit (a: Amount, e: Employee)
Pre:  sometime_past (after (CalculateIncentiveWage (e)))

PersonnelAdministration :: CalculateIncentiveWage (e: Employee)
Pre:  initially (e.HoursWorked = 0)

```

**Figure 8:** Specification on Coordination Level

The pre-condition for the service *Credit* with the parameter values *Amount* and *Employee* states that before a call of the service *Credit* sometimes in the past the service *CalculateIncentiveWage* must have been executed with success. The second part of this specification defines an initial condition of the object *e*. This instance of the structured type *Employee* has to allocate the variable *HoursWorked* with 0. For other temporal operators we refer to [35].

## Conclusion and Outlook

There has been some work done in using the method of standardized specification of business components (see for example [36]) for business domains in the form of case studies. These showed that the concept of multi level contracts about software can be

a valid approach for describing software products. Earlier in this paper it has been noted how we think standardized specification can lead to the evolvement of mature software markets. This is a desirable goal if one looks at the long lasting “software crisis”. We see the current work as a starting point and would like to stimulate further work of the component community.

This paper presented the results of our working group and was intended to stimulate research on an international level. There seem to be some interesting topics in comparing our approach and more semi-formal specification techniques like the Unified Modelling Language (UML) of the Object Management Group (OMG). Another interesting topic could be the specification of non-functional properties or the automated testing for compatibility of business components.

Together with the input of other research communities, these are important steps in the open process of developing a refined methodology for the specification of business components and in general stimulate the usage of formal specification technologies. We think that tool support is also a vital criterion for the acceptance of formal specification. Specification tools for the specification of Business Components currently are being developed. Besides the necessity of institutional frameworks of business components and component reuse to reduce transaction costs of component markets, tool support will be an important factor to establish component markets and make business component specification a mainstream technology.

## References

1. McIlroy, M.D. *Mass Produced Software Components*. in *Software Engineering: Report on a Conference by the NATO Science Committee*. 1986. Brussels: NATO Scientific Affairs Division.
2. Baldwin, C.Y. and K. Clark, *Managing in an age of modularity*. Harvard Business Review, 1997. **75** 5: p. 84-93.
3. Baldwin, C.Y. and K. Clark, *Design Rules: The Power of Modularity*. 2000, London: MIT Press, Cambridge (Mass.).
4. Sanchez, R., *Strategic flexibility in product competition*. Strategic Management Journal 16, 1995: p. 135-159.
5. Sanchez, R. and J.T. Mahoney, *Modularity, flexibility and knowledge management in product and organization design*. Strategic management Journal, 1996. **17**: p. 63-76.
6. Schilling, M.A., *Toward a general modular systems theory and its applications to interfirm product modularity*. Academy of Management Review, 2000. **25**: p. 312-334.
7. Ulrich, K.T., *The role of product architecture in the manufacturing firm*. Research Policy, 1995. **24**: p. 419-440.
8. Sametinger, J., *Software Engineering with Reusable Components*. 1997, Berlin: Springer.
9. Brown, A.W. and K.C. Wallnau, *Engineering of Component-Based Systems*, in *Component-Based Software Engineering: Selected Papers from the*

- Software Engineering Institute*, A.W. Brown, Editor. 1996, IEEE Computer Society Press: Los Alamitos, California. p. 7-15.
10. Orfali, R., D. Harkey, and J. Edwards, *The Essential Distributed Objects Survival Guide*. 1996, New York: John Wiley & Sons.
  11. Hahn, H. and K. Turowski. *General Existence of Component Markets*. in *Sixteenth European meeting on Cybernetics and Systems Research (EMCSR)*. 2002. Vienna.
  12. Fellner, K. and K. Turowski. *Classification Framework for Business Components*. in *Proceedings of the 33rd Annual Hawaii International Conference On System Sciences*. 2000. Maui, Hawaii: IEEE.
  13. Szyperski, C., *Component software: beyond object-oriented programming*. 2 ed. 1998, Harlow: Addison-Wesley.
  14. Meyer, B., *Object-Oriented Software Construction*. 1988, Englewood Cliffs: Prentice Hall.
  15. Meyer, B., *Applying "Design by Contract"*. IEEE Computer, 1992. **25**(10): p. 40-51.
  16. Wilkerson, B. and R.J. Wirfs-Brock, *A Responsibility-Driven Approach*. SIGPLAN Notices, 1989. **24**(10): p. 72-76.
  17. Johnson, R.E. and R.J. Wirfs-Brock, *Surveying Current Research in Object-Oriented Design*. Communications of the ACM, 1990. **33**(9): p. 104-124.
  18. Beugnard, A., et al., *Making Components Contract Aware*. IEEE Computer, 1999. **32**(7): p. 38-44.
  19. OMG, ed. *The Common Object Request Broker: Architecture and Specification: Version 3.0, July 2002*. 2002, OMG: Framingham.
  20. Ehrig, H. and B. Mahr, *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. 1985, Berlin: Springer.
  21. OMG, ed. *OMG Unified Modeling Language Specification, Version 1.4, September 2001*. 2001: Needham.
  22. Hennessy, M., *Algebraic Theory of Processes*. 1988, Cambridge: MIT Press.
  23. Alagar, V.S. and K. Periyasamy, *Specification of Software Systems*. 1998, New York: Springer.
  24. Keller, G., M. Nüttgens, and A.-W. Scheer, *Planungsinseln - Vom Konzept zum integrierten Informationsmodell*. HMD, 1992. **29**(168): p. 25-39.
  25. Rittgen, P. *Objektorientierte Analyse mit EMK*. in *Modellierung betrieblicher Informationssysteme: Proceedings der MobIS-Fachtagung 1999 (MobIS'99)*. 1999. Bamberg: GI-Fachgruppe 5.10.
  26. Jaeschke, P., A. Oberweis, and W. Stucky. *Deriving Complex Structured Objects for Business Process Modelling*. in *Entity-Relationship Approach - ER'94, Proceedings of the 13th International Conference on the Entity-Relationship Approach*. 1994. Manchester: Springer.
  27. Rumbaugh, J., et al., *Object-Oriented Modeling and Design*. 1991, Englewood Cliffs, NY: Prentice Hall.
  28. Coleman, D., *Object-Oriented Development: The Fusion Method*. 1993, Englewood Cliffs: Prentice Hall.
  29. Booch, G., *Object-oriented analysis and design with applications*. 2 ed. 1994, Reading: Addison-Wesley.

30. Jalote, P., *An Integrated Approach to Software Engineering*. 1997, New York: Springer.
31. Ortner, E., *Methodenneutraler Fachentwurf: Zu den Grundlagen einer anwendungsorientierten Informatik*. 1997, Stuttgart: Teubner.
32. Turowski, K., ed. *Standardized Specification of Business Components*. 2002, Gesellschaft für Informatik, Working Group 5.10.3 - Component Oriented Business Application Systems: Augsburg.
33. UNSD, *International Standard Industrial Classification of All Economic Activities*. 2002, United Nations Statistics Division, Third Revision (ISIC, Rev.3), 2002-02-15, <http://unstats.un.org/unsd/>.
34. Basili, V.R. and D.M. Weiss, *A methodology for collecting valid software engineering data*. IEEE Transactions on Software Engineering, November 1984. **SE-10(6)**: p. 728-738.
35. Conrad, S. and K. Turowski, *Temporal OCL: Meeting Specification Demands for Business Components*, in *Unified Modeling Language: Systems Analysis, Design and Development Issues*, K. Siau and T. Halpin, Editors. 2001, Idea Group: Hershey. p. 151-165.
36. Ackermann, J., *Zur Spezifikation der Parameter von Fachkomponenten*, in *Tagungsband des 5. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 5)*, K. Turowski, Editor. 2003: Augsburg. p. 53-160.