Specification of Business Components

Peter Fettke and Peter Loos

Johannes Gutenberg-University Mainz, Chair of Information Systems & Management,
Jakob Welder-Weg 9, D-55099 Mainz, Germany, Phone: +49/6131/39-22734, Fax: -22185,
E-Mail: {fettke|loos}@wiwi.uni-mainz.de, WWW: http://wi.bwl.uni-mainz.de/

Abstract. Component-based software development is a potential reuse paradigm for the future. While the required technologies for a component-style system development are widely available, for instance Sun's Enterprise Java Beans, a problem inhibits the breakthrough of the component paradigm in business application domains: compared to traditional engineering disciplines there is a lack of standardized methods to describe business components. Such a description has to address several aspects: What services are offered and requested by a business component? How can these services be used? Are there any interdependencies between the services of a set of business components? What quality characteristics do the offered services fulfill? And so on. In this paper, we present a holistic approach to specify a business component. This approach consists of seven specification levels which address both technical and business aspects. Furthermore, we show the application of this method by specifying a simple business component that deals with German bank codes.

1 Component-based Business Applications

Enterprise systems are either large standardized off-the-shelf applications, e. g. SAP R/3, Oracle Applications, or BAAN IV, or consist of individual software developments. In contrast, the idea of component-based development (CBD) is to assemble an individual enterprise system from a set of components which are traded on software markets [9; 16; 18]. This approach promises to combine the advantages of standardized off-the-shelf applications and proprietary developments.

CBD can be characterized as a mixture of buying standardized software applications and individual software development. In a CBD scenario, an enterprise that needs a specific business functionality, e. g. an inventory system, can buy required components from different component vendors. The integration of the components can be realized with little effort. The component-based development style offers a great opportunity for small and medium sized enterprises (SME). SMEs normally cannot afford to purchase or even maintain large packaged application systems [19, p. 131].

According to [1; 6, p. 3], the terms component and business components can be defined as:

> A *component* is made up of several (software) artifacts. It is reusable, self-contained, marketable, provides services through a well-defined

interface, hides its implementation, and can be deployed in configurations with other components that are unknown at development time.

The term '(software) artifacts' embraces executable code and its documentation (e. g. comments, diagrams, etc.), an initial description of the component's state (e. g. parameters, initial database setup), specification documents, user documents and test cases. A component is 'reusable' if its integration can be realized easily and without modification of its containing (software) artifacts. Therefore, a customization of a component that is intended by the developer of the component is not considered as a modification. In other words, the customization of components is consistent with the CBD approach. A component is 'self-contained' if its (software) artifacts belong explicitly to the component, so that it can be distinctly differentiated from other components of a component system. This property is a prerequisite for the component's marketability. The characteristic 'marketability' means that components can be traded on component market places. Such a market place can be either an enterprise-wide component repository or an open market place for software components.

A *business component* is a component that provides a certain set of services of a business application domain. Typical business application domains are banking, insurance, retail or manufacturing industry.

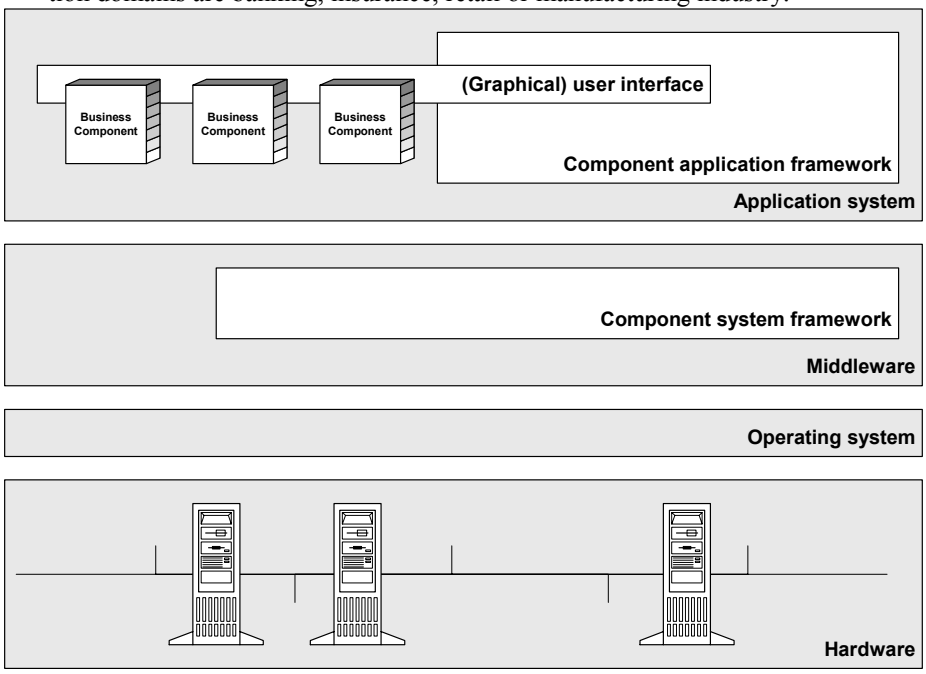

Fig. 1: General architecture of a component-based application system [6, p. 5]

A general architecture of a component-based application system is shown in figure 1 [6, p. 5]. Besides hardware, operating system, and middleware, the architecture consists of a component system framework, a component application framework, and

several business components. The component system framework is part of the middleware and provides application domain independent services such as network communication, transaction processing, object life-cycle management etc. Typical component system frameworks are the Common Object Request Broker Architecture (CORBA), Sun's Enterprise Java Beans, or Microsoft's Distributed Component Object Model (DCOM). The application system framework enables the integration and interoperability of a set of business components on the business level. For instance, it encompasses conflict-solving mechanisms for redundant business functions.

To establish the CBD approach it is necessary to standardize components. A component standard includes both domain standards and methodological standards. A part of a methodological standard is a method to describe components precisely. Such a method is called a specification. A *specification* of a business component is a complete, consistent, and precise description of its outer view.

In this paper we present a specification method for business components. The main contribution of the presented approach is to tie together different well-known and preferably standardized specification notations which are need to specify a business component. So this work provides a means to implement the theory in practice. The proposed specification method distinguishes seven specification levels. We give an overview of the different specification levels in paragraph 2. Paragraphs 3 to 9 discuss each specification level in detail. Therefore, each of these paragraphs describes the purpose and the proposed notation of the corresponding specification level. Furthermore, we discuss in these paragraphs an example specification of a business component that deals with German bank codes [8]. A single bank code is simply a piece of data, e. g. the string or integer "87070000" is the bank code for "Deutsche Bank Chemnitz, Germany". This component can provide responses to queries such as "to which bank does this bank code correspond?" or "is a given bank code valid?". Paragraph 10 summarizes the paper and gives an outlook on further research activities.

The specification method for business components which we present in this paper is the result of the work of the research group "Component-based Development of Business Applications". This research group is a subgroup of the "Gesellschaft für Informatik" (German Informatics Society). A comprehensive description of the specification method is given in [1]. Further information about the research group can be found at the web site "www.fachkomponenten.de" ("Fachkomponenten" is the German term for "business components").

2 Overview of the Specification Levels

According to [4; 20; 21], it is useful to specify a business component on different levels. Each level focuses on a specific aspect of a business component specification and addresses different development roles such as reuse librarian, reuse manager, component developer etc. [2, pp. 337-340]. The proposed method divides the specification of a business component into seven levels (figure 2).

Various notations are used on all specification layers. We prefer a formal notation as a primary notation because of its precision and consistency. Furthermore, we also

introduce on some specification levels a secondary notation which may be semi-formal or informal. The secondary specification improves the specification comprehensibility and can be considered as a supplementary specification for people not used to formal specifications.

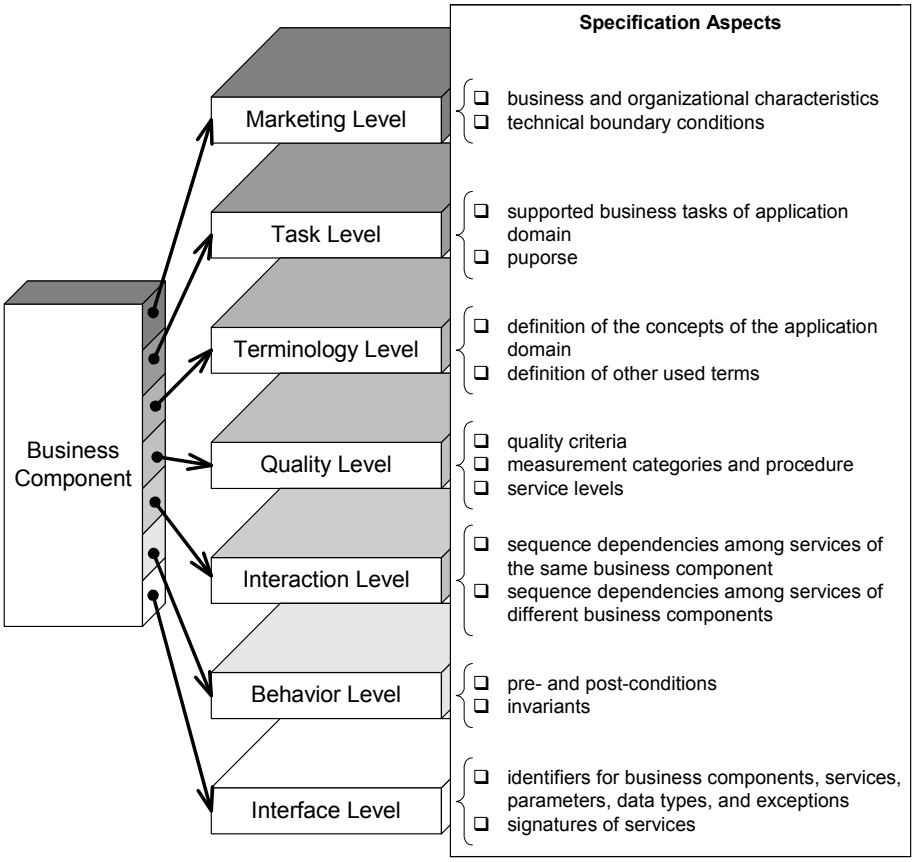

Fig. 2: Specification levels and specification aspects [1, p. 4]

3 Interface Level

The interface level describes the services that are offered by a business component on a technical level. For that purpose the services, public attributes, public variables or constants, and public data types are named, the signature of every service is defined, and possible error states and exceptions are declared. Not only the offered services, but also the services required by the business component to fulfill a smooth operating of the business component are specified. In other words, a business component can be

logically considered both as a server that offers services and as a client that requests services.

The OMG Interface Definition Language (IDL) [13] is proposed as a notation for the interface level. This notation is suitable for describing the mentioned specification aspects. Moreover, it is a widely used and well-known notation in industry and research.

```
interface BankCodes
{
  typedef integer BankCode;

  struct Bank
  {
    BankCode bankCode;
    string   bankName;
    string   city;
  }

  sequence <Bank> ListOfBanks;

  exception UndefinedBank();

  boolean isValidBankCode(in BankCode bankCode);
  boolean isValidBankName(in string   bankName);
  boolean isValidBank    (in BankCode bankCode,
                          in string   bankName);
  string  searchBankCode (in string   bankName) raises
                                      (UndefinedBank);
  string  searchBankName (in BankCode bankCode) raises
                                      (UndefinedBank);
  ListOfBanks searchBanksWithWildcard(
                          in string   bankName);
  ListOfBanks searchBanksWithWildcardAndCityWildcard(
                          in string   bankNameWildcard,
                          in string   cityWildcard);
};

interface extern
{
};
```

Fig. 3: Example specification of the interface level

The interface specification of the business component "BankCodes" is given in figure 3. This business component offers seven services. Furthermore, some public data types (BankCode, Bank, and ListOfBanks), and an exception (UndefinedBank) are declared. This business component does not require services of other business components. Therefore, its extern interface is empty.

4 Behavior Level

This level describes the behavior of the services which are offered by a business component. This improves the level of confidence of using the business component. Whereas the interface level primarily addresses syntactic issues of using a business component, the behavior level specifies the behavior of services of business components in general and especially in worst case scenarios. For that purpose pre- and post-conditions of using a service and possibly invariants are defined.

The Object Constraint Language (OCL) is proposed as a notation for the behavior level. Formerly, this notation was not part of the Unified Modeling Language (UML), but in the meantime the OMG added the OCL to the UML standard [14]. As a secondary notation on the behavior level, each OCL expression may be annotated with comments.

Figure 4 shows the specification of a business component on the behavior level. First, the context of the specification must be defined. The context is declared by an underline. For instance, the context of the first expression is the whole business component "BankCodes". The second expression refers to the service "searchBankCode" of the business component "BankCodes".

The first expression specifies that each bank which is managed by the business component must have a bank code greater than zero. Furthermore, the name of each bank must not be empty. The second expression specifies that the service "searchBankCode" may only be called if a bank with the given name exists. Similar expressions may be defined for the other services of the business component.

```
(1)
BankCodes
  self.ListOfBanks->forAll(b:Bank | b.bankCode > 0)
  self.ListOfBanks->forAll(b:Bank | b.bankName <> '')

(2)
BankCodes::searchBankCode(name : bankname): bankCode
  pre : self.ListOfBanks->exists(b:Bank | b.bankName =
                                          bankname)
```

Fig. 4: Example specification of the behavior level

5 Interaction Level

Sometimes it is necessary to define sequences in which services of a business component are allowed to be used. For instance, the service "send reminder of payment" may only be called after the service "charge to the customer's account". The interaction level aims to specify these dependencies among the services of business components. A specific dependency can exist among the services which belong to the same busi-

ness component (intra-component dependency) or between different business components (inter-component dependency).

A temporal logic may be used to express such dependencies. The authors of [5] propose an extension of the OCL with temporal operators. This proposal is used as a notation for the interaction level. Because the proposed notation is just an extension of the OCL, its advantage is a smooth integration of the behavior and interaction levels.

The following temporal operators can be used (A, B are Boolean terms):

- *sometime_past* A: A was true at one point in the past.
- *always_past* A: A was always true in the past.
- A *sometime_since_last* B: A was true sometime in the past since the last time B was true.
- A *always_since_last* B: A was always true since the last time B was true.
- *sometime* A: A will be true sometime in the future.
- *always* A: A will be true always in the future.
- A *until* B: A is true until B will be true in the future.
- A *before* B: A will be true sometime in the future before B will be true in the future.
- *initially* A: At the initial state A is true.

As a secondary notation on the interaction level, every OCL expression may be annotated with comments.

The business component "BankCodes" is rather simple, its services have no dependencies on other services (cf. paragraph 3). But its services may be used by other business components, for instance a business component "BankTransfer" offers a service to execute a payment order. This service may only be called after it is verified that the recipient's bank details are valid. The first constraint in figure 5 specifies this dependency. Alternatively, this verification may be assured after the payment order is accepted. This dependency is specified by the second constraint in figure 5.

```
(1)
BankTransfer::ExecutePayment(order : PaymentOrder)
  pre : sometime_past
            (BankCodes::isValidBank(order.recipientBank))

(2)
BankTransfer::AcceptPayment(order : PaymentOrder)
  post : sometime
            (BankCodes::isValidBank(order.recipientBank))
```

Fig. 5: Example specification of the interaction level

6 Quality Level

The specification levels mentioned above focus on the functional characteristics of business components. In addition, it is necessary to define non-functional qualities of a business component. This is the purpose of the specification of the quality level.

Such characteristics include availability, error recovery time, throughput time, response time etc. and can be classified as static or dynamic characteristics. Static characteristics, e. g. size of a component, can be measured at the build-time of a business component; dynamic characteristics, e. g. the response time of a called service, can only be measured during the run-time.

Various quality parameters depend on the boundary conditions of the runtime environment (e. g. main memory size, processor type, database management system (DBMS) etc.). Therefore, it is essential to define these boundary conditions accurately. If the boundary conditions are only fuzzy, then it is impossible to measure objective quality criteria. It must be pointed out, that there is a wide range of possible boundary conditions. Some business components may rely on network capacity, others on the DBMS or the processor type etc. Consequently, no universally valid boundary conditions can be defined.

Instead, a general procedure to define and specify quality criteria is introduced. The procedure consists of four steps (figure 6).

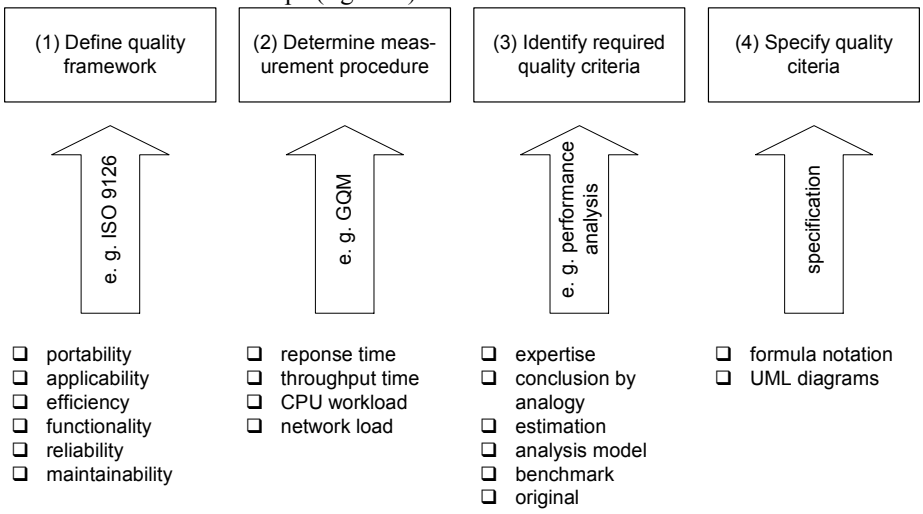

Fig. 6: General procedure to define and specify quality criteria [1, p. 13]

(1) Define quality framework: The term quality of a business component can be defined in various ways. To that effect, a quality framework has to be introduced to define the concept quality. For this so-called "Factor Criteria Metrics" models such as the ISO 9126 may be used [3, pp. 255-276].
(2) Identify required quality criteria: At the next step, the specific quality criteria will be stated. For instance, the Goal/Question/Metric Method may be used for this [17].
(3) Determine measurement methods: The identified quality criteria have to be quantified by a specific measurement method. Typical methods are expertise, conclusion by analogy, benchmark, estimation, analysis model, and original.

(4) Specify quality criteria: As the last step, the specific quality criteria of a business component can be defined. For this, no specific notation is proposed. Possible notations are formula notation, or UML diagrams.

Because of the mentioned problems, it is rather difficulty to depict a simple example of a quality specification. Nevertheless, a specification of the quality level is sketched out. Starting point of the measurement is the following reference environment:

- processor type: Intel Pentium III 866 MHz
- main memory: 1 GB RAM
- operation system: Windows NT 4.0
- database management system: Oracle 8i
- component system framework: Brokat Twister 2.3.5

The quality of the service "isValidBankCode" is specified in figure 7. The service is called with randomly numbers with uniform distribution which lie in the interval "10000000" to "99999999".

Quality criteria	Specification
throughput	1.8 s (workload: 1000 requests)
response time	18 ms
response time distribution	0.0324 ms
availability	not available
error recovery time	not available

Fig. 7: Example specification of quality the level

7 Terminology Level

Every specification level uses various concepts that have a specific, usually not standardized denotation, e. g. identifiers on the interface, behavior, and interaction level, or tasks on the task level. Therefore, it is necessary to define each concept explicitly. This is the purpose of the terminology level. The terminology level can be viewed as a glossary of every concept that is needed to understand the business component specification.

The requirements of the terminology level can be fulfilled by so-called standardized business languages (SBLs) [10; 15]. SBLs use explicitly defined patterns to build sentences, and their vocabulary is based on a reconstructed colloquial language. Thus, it is possible to clarify the use of synonyms, homonyms, and other language defects.

SBLs use various definition methods:

- explicit definition of a term based on already defined terms,
- definition of relationships between concepts such as is-broader, is-narrower, is-related etc., and
- introduction of new concepts by examples and counter-examples (to overcome the problem of the beginning of the definition process).

In the following, some sample patterns for building sentences are described (A, B are objects in a general meaning):

- abstract relationship: A is B
- component relationship: A compounds / is part of B
- sequence relationship: A happens before / after / concurrent B

A specification of the terminology level is shown in figure 8. This simple example uses just the abstract relationship pattern.

Concept	Definition
bank code	A bank code is an eight-digit number that identifies a German financial institution. It is also used for transactions between a financial institution and the "Deutsche Bundesbank" (Central Bank of the Federal Republic of Germany). Examples: "87070000", "12345678", counter-examples "1234567", "Deutsche Bank" Synonyms: BC, bank number
bank name	A bank name is an identifier of a financial institution according to the field "brief description of a financial institution office" in the file "SATZ188.doc" (source: Deutsche Bundesbank, www.bundesbank.de). Example "Deutsche Bank Chemnitz, Germany", counter-example: "87070000"
Response time	The response time is the time period between the call of a business component's service and its termination.

Fig. 8: Example specification of the terminology level

8 Task Level

By definition, a business component supports the execution of a set of various business tasks. The purpose of the task level is to specify which business tasks are supported by the business component. This information in combination with the marketing layer describes the application domain of a business component.

In contrast to the technical-oriented interface level, the task level provides a conceptual description of a business component. Therefore, as on the terminology level, a SBL is proposed as a specification notation (cf. paragraph 7).

The specification of the task level of the business component "BankCodes" is shown in figure 9.

Task	Description
verify bank code	This task verifies if a given bank code is valid or if a given bank code corresponds to a given bank name.
look up bank code	This task looks up the bank code for a given bank name.
look up bank	This task looks up the bank name for a given bank code.

Fig. 9: Example specification of the task level

9 Marketing Level

The purpose of the marketing level is to ensure the efficient handling of business components from a business or organizational perspective. The specification of the marketing level is especially needed to trade a component on a component market place. It covers three groups of characteristics:

- business and organization characteristics,
- technical boundary conditions, and
- miscellaneous characteristics.

As a notation a tabular form is proposed (figure 10). These conventions are used:

- Each table entry names one attribute which is typed in **bold** letters.
- If the attribute is optional, it is enclosed in square brackets. Example: [optional attribute]
- If the attribute may be specified more than once, it is enclosed in curly brackets. Example: {repeatable attribute}

The authors of [7] proposed an alternative approach to specify this level that is based on the Extensible Markup Language (XML).

Figure 11 shows the specification of the marketing level of the business component "BankCodes".

Name The name of the business component.
Identifier A unique identifier to identify the business component.
Version This attribute defines version and release of the business component.
Branch of Economic Activity This attribute describes the application domain of the business component from an economical perspective. The International Standard Industrial Classification of All Economic Activities [22] is used, e. g.: manufacturing, retail trade, or financial intermediation.
{Domain} This attribute describes the application domain of the business component from a functional perspective. Possible values are: research and development, sales, procurement, inventory, production, delivery, after sales services, finance, accounting, human resource, facility management [11; 12].
Scope of Supply This attribute specifies all (software) artifacts which belong to the business component.
{Component Technology} This attribute specifies the component technology.

Fig. 10: Specification notation for the marketing level (part 1 of 2)

{System Requirements} This attribute specifies the system environment that is needed by the business component, e. g. processor type, memory size, secondary memory size, operating system and its version, component system and application framework and their versions, etc.
[Manufacturer] This attribute describes the manufacturer of the business component.
[Contact Person] This attribute names a contact person.
[Contractual Basis] This attributes describes modalities to buy the component, e. g. costs per license, terms of license, terms of payment etc.
[Miscellaneous] This attribute allows definition of further characteristics of a business component that may be relevant to a potential user.

Fig. 10: Specification notation for the marketing level (part 2 of 2)

Name BankCodes
Version V 1.0
Branch of Economic Activity Independent
Domain Accounting
Scope of Supply bankcodes.jar: implemented Java-Classes bankcodes.tws: IDL specification of the component create_db.sql: SQL script to setup the database blz0010pc.txt: original source of bank codes data offered by the "Deutsche Bundesbank" (source: http://www.bundesbank.de/) script_sampledata.pl: Perl script to convert the original source of bank codes to a file containing SQL statements bankcodestests.jar: implemented test cases for the business component
Component Technology Brokat Twister 2.3.5
System Requirements processor type: x86 main memory size: 512 MB operating system: Windows NT 4.0, SP 3 database management system: Oracle 8i component system framework: Brokat Twister 2.3.5
Manufacturer Chemnitz University of Technology, Information Systems & Management, D-09107 Chemnitz

Fig. 11: Example specification of the marketing level

10 Summary and Further Work

This paper proposes a method for specifying business components. According to this approach a business component is described on seven levels which cover both technical and business aspects. Thus, this approach can be characterized as a holistic specification of business components.

In the near future, we will gain more experience with applying this method to various application domains. This may lead to domain standards in the long-term. Furthermore we have to point out that this proposed approach is primarily a notation standard. To establish this standard it is necessary – among other tasks – to develop a procedure model for component specification that is based on this notation standard.

References

1. Ackermann, J.; Brinkop, F.; Conrad, S.; Fettke, P.; Frick, A.; Glistau, E.; Jaekel, H.; Kotlar, O.; Loos, P.; Mrech, H.; Raape, U.; Ortner, E.; Overhage, S.; Sahm, S.; Schmietendorf, A.; Teschke, T.; Turowski, K.: Vereinheitlichte Spezifikation von Fachkomponenten - Memorandum des Arbeitskreises 5.10.3 Komponentenorientierte betriebliche Anwendungssysteme. http://wi2.wiso.uni-augsburg.de/gi-memorandum.php.htm, access date: 2002-05-10. Augsburg 2002.
2. Allen, P.; Frost, S.: Component-Based Development for Enterprise Systems - Applying the Select Perspective. Cambridge 1998.
3. Balzert, H.: Lehrbuch der Software-Technik - Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg, Berlin 1998.
4. Beugnard, A.; Jézéquel, J.-M.; Plouzeau, N.; Watkins, D.: Making Components Contract Aware. In: IEEE Computer 32 (1999) 7, pp. 38-45.
5. Conrad, S.; Turowski, K.: Vereinheitlichung der Spezifikation von Fachkomponenten auf der Basis eines Notationsstandards. In: J. Ebert; U. Frank (Eds.): Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik - Beiträge des Workshops "Modellierung 2000", St. Goar, 5.-7. April 2000. Koblenz 2000, pp. 179-194.
6. Fellner, K. J.; Turowski, K.: Classification Framework for Business Components. In: R. H. Sprague (Eds.): Proceedings of the 33rd Hawaii International Conference on System Sciences. Maui, Hawaii 2000
7. Fettke, P.; Loos, P.: Ein Vorschlag zur Spezifikation von Fachkomponenten auf der Administrations-Ebene. In: K. Turowski (Eds.): Modellierung und Spezifikation von Fachkomponenten: 2. Workshop im Rahmen der vertIS (verteilte Informationssysteme auf der Grundlage von Objekten, Komponenten und Agenten) 2001, Bamberg, Deutschland, 05. Oktober 2001. Bamberg 2001, pp. 95-104.
8. Fettke, P.; Loos, P.; Tann von der, M.: Eine Fallstudie zur Spezifikation von Fachkomponenten eines Informationssystems für Virtuelle Finanzdienstleister - Beschreibung und Schlussfolgerungen. In: K. Turowski (Eds.): Modellierung und Spezifikation von Fachkomponenten: 2. Workshop im Rahmen der vertIS (verteilte Informationssysteme auf der Grundlage von Objekten, Komponenten und Agenten) 2001, Bamberg, Deutschland, 05. Oktober 2001. Bamberg 2001, pp. 75-94.
9. Griffel, F.: Componentware - Konzepte und Techniken eines Softwareparadigmas. Heidelberg 1998.
10. Lehmann, F. R.: Normsprache. In: Informatik Spektrum 21 (1998) 6, pp. 366-367.

11. Mertens, P.: Integrierte Informationsverarbeitung 1 - Operative Systeme in der Industrie. 13. ed., Wiesbaden 2001.
12. Mertens, P.; Griese, J.: Integrierte Informationsverarbeitung 2 - Planungs- und Kontrollsysteme in der Industrie. 8. ed., Wiesbaden 2000.
13. OMG: The Common Object Request Broker: Architecture and Specification: Version 2.5. Framingham 2001.
14. OMG: Unified Modeling Language Specification: Version 1.4. Needham 2001.
15. Ortner, E.: Methodenneutraler Fachentwurf - Zu den Grundlagen einer anwendungsorientierten Informatik. Stuttgart, Leipzig 1997.
16. Sametinger, J.: Software Engineering with Reusable Components. Berlin et al. 1997.
17. Solingen, R. v.; Berghout, E.: The Goal/Question/Metric Method - A Practical Guide for Quality Improvement of Software Development. London et al. 1999.
18. Szyperski, C.: Component Software - Beyond Object-Oriented Programming. Harlow, England, et al. 1999.
19. Turowski, K.: Establishing Standards for Business Components. In: K. Jakobs (Eds.): Information Technology Standards and Standardisation: A Global Perspective. Hershey 2000, pp. 131-151.
20. Turowski, K.: Fachkomponenten - Komponentenbasierte betriebliche Anwendungssysteme. Habil.-Schr., Magdeburg 2001.
21. Turowski, K.: Spezifikation und Standardisierung von Fachkomponenten. In: Wirtschaftsinformatik 43 (2001) 3, pp. 269-281.
22. United Nations (Eds.): International Standard Industrial Classification of All Economic Activities, Third Revision, (ISIC, Rev.3). http://unstats.un.org/unsd/cr/family2.asp?Cl=2, access date: 2002-07-01. 1989.