UNIFYING THE SPECIFICATION OF BUSINESS COMPONENTS

Peter Fettke
Information Systems and Management, University Mainz, D-55099 Mainz, Germany
Peter Loos
Information Systems and Management, University Mainz, D-55099 Mainz, Germany
Correspondence Email: fettke@isym.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 addressing 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.

Keywords: Component Engineering, Specification, Service-Architecture, Web-Services

1. INTRODUCTION

Enterprise systems are either large standardized off-the-shelf applications like 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 [14, 16]. 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 developing individual software. In a CBD scenario, an enterprise that needs particular business functionality, e.g. an inventory system, can buy required components from different component vendors and integrate them with little effort. The CBD offers an opportunity for small and medium sized enterprises (SME) which normally cannot afford to purchase and maintain large packaged application systems [17, p. 131].

According to [5, p. 3], the terms component and business components can be defined as follows: 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. A business component is a component that provides a certain set of services from a business application domain. Typical business application domains are banking, insurance, retail or manufacturing industry. 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 it can be integrated easily and without modification of its (software) artifacts. 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 unambiguous differentiated from other components of a software system. This property is a prerequisite for the component's marketability. The characteristic 'marketability' means that components can be traded on component market places.

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 needed to specify a business component. Furthermore, we develop a meta model for the proposed specification framework that facilitates the communication about the framework, supports teaching the framework, and simplifies the implementation of the framework. So, this work provides means to implement theory in practice.

There are many approaches to software reuse and software repositories in general (for an overview see [10]). However, we are just aware of three approaches which addresses the specification of (business) components in particular [3, 6, 8]. Compared to these existing approaches, the proposed method is more comprehensive because it comprises both business and technical aspects.

The remainder of the paper proceeds as follows. The proposed specification method is introduced in section 2. Here, an example specification of a business component that deals with German bank codes will be discussed. A bank code is simply a unique identifier, 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?". A meta model for the proposed specification method is presented in Section 3. Finally, section 4 summarizes the paper and presents directions for further research.

The specification method for business components that 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. SPECIFYING BUSINESS COMPONENTS

2.1. Overview

We postulate, that a method for specifying business components is necessary (but not sufficient) to establish a component community. According to [3], it is useful to specify a business component on different levels. Our approach uses seven specification levels (cf. figure 1). 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, component vendor etc. [2, pp. 337-340].

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 that 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. The subsequent paragraphs of this section describe each specification level in more detail.

2.2. 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.

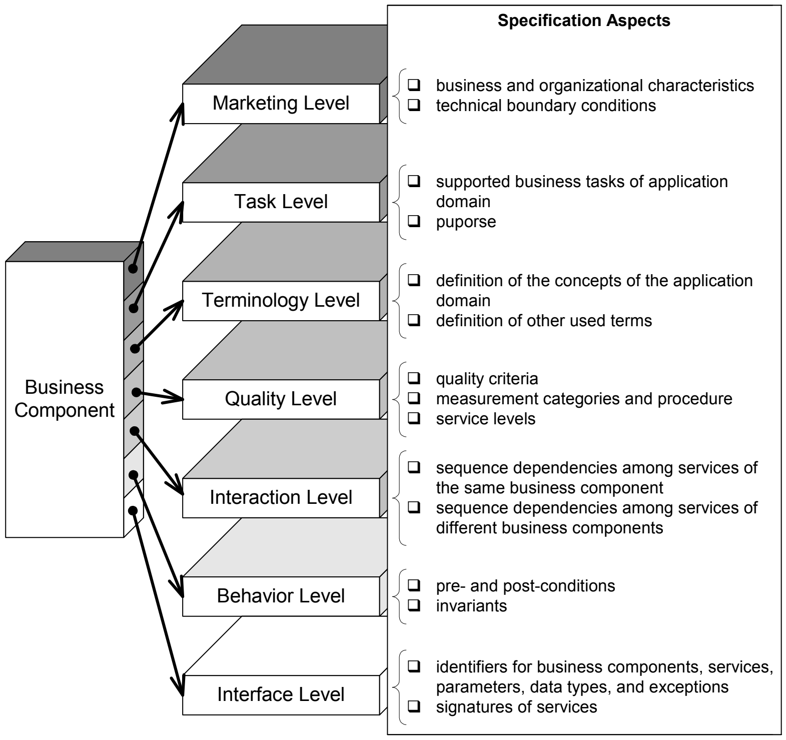
Specification Aspects
Business Component
Marketing Level
business and organizational characteristics
technical boundary conditions
Task Level
supported business tasks of application domain
puporse
Terminology Level
definition of the concepts of the application domain
definition of other used terms
Quality Level
quality criteria
measurement categories and procedure
service levels
Interaction Level
sequence dependencies among services of the same business component
sequence dependencies among services of different business components
Behavior Level
pre- and post-conditions
invariants
Interface Level
identifiers for business components, services, parameters, data types, and exceptions
signatures of services

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

The OMG Interface Definition Language (IDL) [11] 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.

The interface specification of the business component "BankCodes" is given in figure 2. 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.

2.3. Behavior Level

This level describes the behavior of the services that 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 [12]. As a secondary notation on the behavior level, each OCL expression may be annotated with comments.

```
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 {};
```

Figure 2: Example specification of the interface level

Figure 3 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 that 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)
```

Figure 3: Example specification of the behavior level

2.4. 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 that belong to the same business 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 [4] 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 3.2). 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 4 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 4.

This specification level addresses aspects of the communication between components (component-to-component specification). But note, the definition of such communication dependency need not rely on one specific business component as shown in the example. Instead the specification dependency can refer to an indefinite business component. In this case, the key word "extern" is used as a component identifier.

```
(1) BankTransfer::ExecutePayment(order : PaymentOrder)
    pre : sometime_past
            (BankCodes::isValidBank(order.recipientBank))
(2) BankTransfer::AcceptPayment(order : PaymentOrder)
    post : sometime
            (BankCodes::isValidBank(order.recipientBank))
```

Figure 4: Example specification of the interaction level

2.5. 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 5).

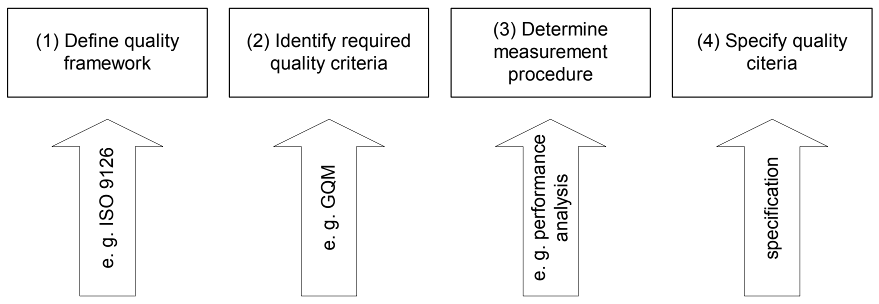
(1) Define quality framework
(2) Identify required quality criteria
(3) Determine measurement procedure
(4) Specify quality citeria
e. g. ISO 9126
e. g. GQM
e. g. performance analysis
specification

Figure 5: 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.
(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 [15].
(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 6. 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
error recovery time	not available

Figure 6: Example specification of quality the level

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". Examples: "87070000", "12345678", 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.

Figure 7: Example specification of the terminology level

2.6. 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) [13]. 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. A SBL allows an explicit specification of the conceptualizations of all terms used on every specification level. Therefore, it is comparable to an ontological approach [7].

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 7. This simple example uses just the abstract relationship pattern.

2.7. 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 3.6).

The specification of the task level of the business component "BankCodes" is shown in figure 8.

Task	**Description**
verify bank code	This task verifies 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.

Figure 8: Example specification of the task level

2.8. 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.

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 [18] 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. It is based on a reference model constructed by [9]. Possible values are: research and development, sales, procurement, inventory, production, delivery, after sales services, finance, accounting, human resource, facility management.
Scope of Supply: This attribute specifies all (software) artifacts that belong to the business component.
{Component Technology}: This attribute specifies the component technology.
{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.

Figure 9: Specification notation for the marketing level

As a notation a tabular form is proposed (figure 9). 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}

Figure 10 shows the specification of the marketing level of the business component "BankCodes".

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 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, D-09107 Chemnitz

Figure 10: Example specification of the marketing level

3. SPECIFICATION META MODEL

This Section presents a meta model for the specification framework developed in Section 2. A meta-model attempts to adequately model aspects of a given modeling language.

Figure 11 depicts a meta model for the business component specification framework. We use the Entity-Relationship model as a meta modeling language. The specification of a business component (entity type "Business Component Specification") is decomposed into seven specification layers (entity types "Interface Level", "Behavior Level" etc."). The main constructs of each specification level are specified in more detail. For instance, on the terminology level, several concepts can be defined (entity type "Concept"). Each concept is exactly related to one terminology specification. It is possible to specify several relationships between the defined concepts (entity type "Relationship (concept)"), whereas each relationship between concepts is of a particular relationship type (entity type "Relationship type (concept)"). For example, the concept "customer" is more specific ("relationship type") than the concept "business partner".

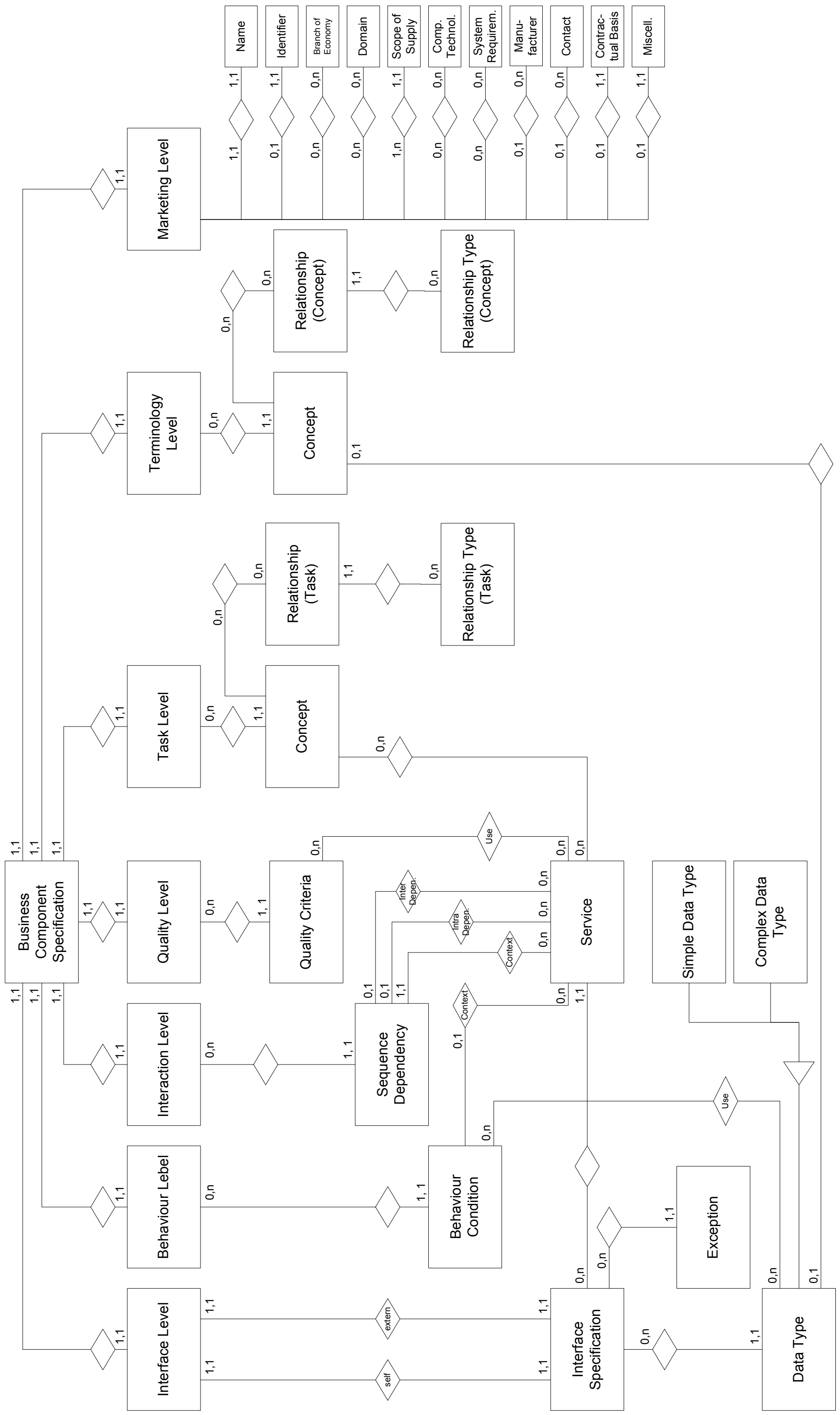

Figure 11: Meta model of the business component specification framework

4. 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 covering both technical and business aspects. Thus, this approach can be characterized as a holistic specification of business components.

There are several areas of further research. First, more experience is needed to apply the proposed specification method to various application domains. To gain more experience, empirical research methods (case studies, action research etc.) can be applied. This work validates the proposed method. Second, we have to point out that this approach is primarily a notation standard. To establish this standard it is necessary – among other tasks – to develop a procedural model for component specification that is based on this notation standard. Third, we are preparing an ontological evaluation [19] of this method to introduce an ontological – hopefully more precise – definition of the specification framework's constructs. In the long term, we believe, the mentioned areas of further work may lead to standards for different application domains based on a sound specification method, e.g. standard business components for manufacturing, retail, banking, insurance etc.

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.: Standardized Specification of Business Components. http://wi2.wiwi.uni-augsburg.de/gi-memorandum.php.htm.

[2] Allen, P.; Frost, S.: Component-Based Development for Enterprise Systems - Applying the Select Perspective. Cambridge 1998.

[3] Beugnard, A.; Jézéquel, J.-M.; Plouzeau, N.; Watkins, D.: Making Components Contract Aware. In: IEEE Computer 32 (1999) 7, pp. 38-45.

[4] Conrad, S.; Turowski, K.: Vereinheitlichung der Spezifikation von Fachkomponenten auf der Basis eines Notationsstandards. In: J. Ebert; U. Frank (Ed.): Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik - Beiträge des Workshops "Modellierung 2000", St. Goar, 5.-7. April 2000. Koblenz 2000, pp. 179-194.

[5] Fellner, K. J.; Turowski, K.: Classification Framework for Business Components. In: R. H. Sprague (Ed.): 33rd Hawaii International Conference on System Sciences. Maui, Hawaii 2000

[6] Fischer, B.: Specification-Based Browsing of Software Component Libraries. In: Journal of Automated Software Engineering 7 (2000) 2, pp. 179-200.

[7] Gruber, T. R.: Towards principles for the design of ontologies used for knowledge sharing. In: International Journal of Human-Computer Studies 43 (1994) 5/6, pp. 907-928.

[8] Han, J.: An Approach to Software Component Specification. Proceedings of 1999 International Workshop on Component Based Software Engineering. Los Angeles, USA 1999

[9] Mertens, P.: Integrierte Informationsverarbeitung 1 - Operative Systeme in der Industrie. 13. ed., Wiesbaden 2001.

[10] Mili, H.; Mili, A.; Yacoub, S.; Addy, E.: Reuse-Based Software Engineering - Techniques, Organizations, and Controls. 2001.

[11] OMG: The Common Object Request Broker: Architecture and Specification: Version 2.5. Framingham 2001.

[12] OMG: Unified Modeling Language Specification: Version 1.4. Needham 2001.

[13] Ortner, E.: Methodenneutraler Fachentwurf - Zu den Grundlagen einer anwendungsorientierten Informatik. Stuttgart, Leipzig 1997.

[14] Sametinger, J.: Software Engineering with Reusable Components. Berlin et al. 1997.

[15] Solingen, R. v.; Berghout, E.: The Goal/Question/Metric Method - A Practical Guide for Quality Improvement of Software Development. London et al. 1999.

[16] Szyperski, C.: Component Software - Beyond Object-Oriented Programming. 2nd ed., London 2002.

[17] Turowski, K.: Establishing Standards for Business Components. In: K. Jakobs (Ed.): Information Technology Standards and Standardisation: A Global Perspective. Hershey 2000, pp. 131-151.

[18] United Nations (Ed.): International Standard Industrial Classification of All Economic Activities, Third Revision, (ISIC, Rev.3). http://unstats.un.org/unsd/cr/family2.asp?Cl=2, 1989.

[19] Wand, Y.; Weber, R.: On the ontological expressiveness of information systems analysis and design grammars. In: Journal of Information Systems 3 (1993) 4, pp. 217-237.