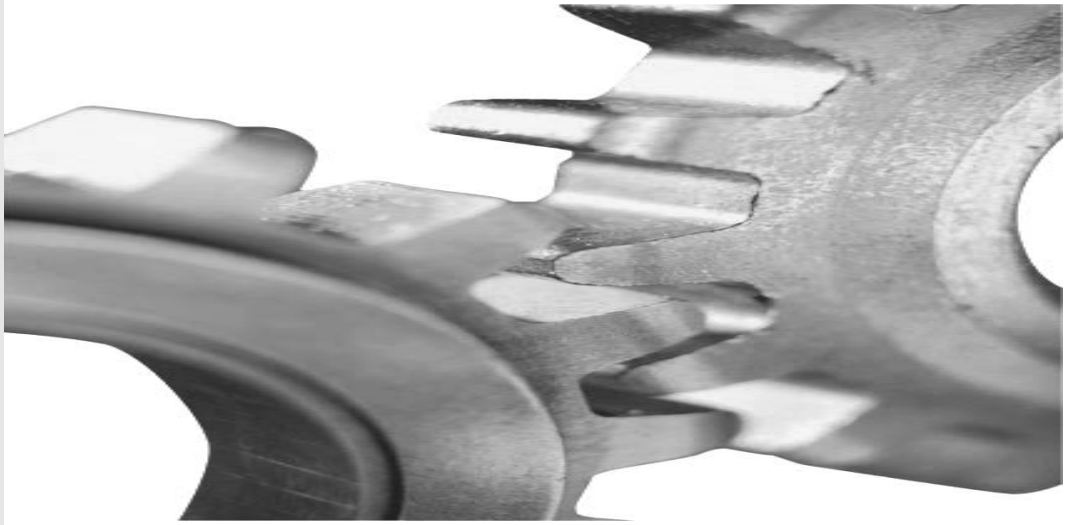
Klaus Turowski
(Editor)

Jörg Ackermann
Frank Brinkop
Stefan Conrad
Peter Fettke
Andreas Frick
Elke Glistau
Holger Jaekel
Otto Kotlar
Peter Loos
Heike Mrech
Erich Ortner
Ulrich Raape
Sven Overhage
Stephan Sahm
Andreas Schmietendorf
Thorsten Teschke
Klaus Turowski

Standardized Specification of Business Components

Memorandum of the working group 5.10.3
Component Oriented Business Application System

February 2002



Gesellschaft für Informatik
Working Group 5.10.3
Component Oriented Business Application Systems

Preface

Traditional engineering disciplines are generally characterized by accepted methodical standards for notation, nomination and the use of measurements to specify the relevant result of construction in order to facilitate reusing them in a different context. Hence, only by looking at a constructional drawing, an engineer is able to understand the constructive and material properties of any work piece even though it is new to him. Furthermore, he can reuse these engineering results as a solution for a problem he is facing in his own context, which might be considerably different.

In the field of software engineering and in particular in the field of development of business application systems, the situation looks completely different: Component strategies based on accepted standards are - if at all - only successful in areas that are independent of a specific application domain (such as middleware). However, the above mentioned obligatory and accepted methodical standards do not exist. Thus, the change to a software development process which is based on the reuse of existing solutions is restrained and in this way the efficiency potential of the industrial production process remains unexploited for the production of software solutions. For example, there is an ongoing discussion how to specify software components, which notation should be used and in general, which separate units should be specified. It is evident that this situation makes it impossible to teach a uniform and standardized theory for the construction of software components at university or elsewhere. Furthermore the practical advantages for the process of software development remain unexploited, as well.

Concerning the mentioned problem a first workshop *Modeling and Specification of Business Components* took place October 12th, 2000 in Siegen / Germany, hosted by the working group 5.10.3 *Component Oriented Business Application Systems* by the German Business Information Systems research community (GI) in the context of the conference *Modelling of Business Information Systems (MoBIS).* It was decided to compose a draft dealing with the standardization of the specification of business components. The participants were called to contribute to the memorandum of the same name which finally in this first consolidated version.

On the way, eleven intermediate papers were written, discussed among the authors and presented to the public – the last during the second workshop Modelling and Specification of Business Components. Organized by the workgroup 5.10.3, it took place October 5th, 2001 in Bamberg / Germany during the conference Distributed Information Systems based on Objects, Components and Agents (vertVIS 2001).

Last changes were made during the workshop Standardization for Specification of Business Components which was hosted by the working group 5.10.3 together with the university of Augsburg with friendly support by SAP AG in Walldorf / Germany on December 3rd/4th, 2001.

An electronic version of this memorandum is available on www.fachkomponenten.de. Furthermore, the site contains examples and case studies on specifications of business components according to this memorandum, ongoing projects, references to lectures and lecture materials concerning the specification of business components as well as coming events by the working group 5.10.3.

To finish, we thank all participants and contributors for their engagement, time and articles, which have made this memorandum possible. Furthermore, many thanks to all those whose organizational support was crucial to achieve this work, in particular Andreas Krammer and Moritz Weizmann, who provided the translation.

Augsburg, February 2002

Klaus Turowski

Index

Abbreviations

CORBA	Common Object Request Broker Architecture
COTS	Commercial off-the-shelf
CPU	Central Processing Unit
DBMS	Database management system
DTD	Document Type Definition
EJB	Enterprise Java Bean
FCM	Factor Criteria Metrics Model
GQM	Goal/Question/Metric
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
OCL	Object Constraint Language
OMG	Object Management Group
RDF	Resource Description Framework
SQL	Structured Query Language
UML	Unified Modeling Language
WSDL	Web Service Description Language
XML	Extensible Markup Language

Figures

1 Compositional Reuse and Basic Terms

Combining off-the-shelf software components offered by different vendors to customer-individual business application systems is a goal that has been followed-up for a long time [McIl1968]. By achieving this goal, advantages of individually programmed software with those of standard, off-the-shelf software could come together. Guiding model as an ideal future scenario is a compositional, plug-and-play like reuse of black-box components whose implementation remains invisible to users and which can be traded on component markets to be joined to customer- individual business application systems.

Corresponding to our guiding model, a company, which e.g. needs new software for stock keeping, could buy a suitable software component on the component market and further integrate it into its business application system with little effort. The relevant market could be an open market (defined by the term commercial-off-the-shelf (COTS) Components) or a company's internal repository.

According to [FeRT1999, S. 34] the terms *component* and *business component* are defined as follows:

> A *component* consists of different (software) artifacts. 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 *(software) artifact* can be executable code, included graphics, texts etc., data that describes the initial state of a component, such as parameters, as well as specification and user documentation and (automatic) tests.

The criterion of *reusability* is fulfilled, when a component can be integrated in other software systems without modification of its (software) artifacts, apart from intended modification possibilities such as parameters.

The criterion of *being self-contained* is fulfilled, when its parts (i.e. the (software) artifacts) can be assigned unequivocally in order to distinguish it as a unit from other parts of the system. Consequently, being self-contained is a precondition for being marketable.

Marketable means that in principle components can be identified as a separate good, so that they can be traded on an open as well as company internal market.

> A *business component* is a component that offers a certain set of services of a given *business* domain.

A precondition to component based development of application systems by using business components is a stable component model. Additionally it seems suitable to distinguish different kinds of components. Fig. 1 and 2 show an illustration of a distinction between component types and a meta-schema for the component model.

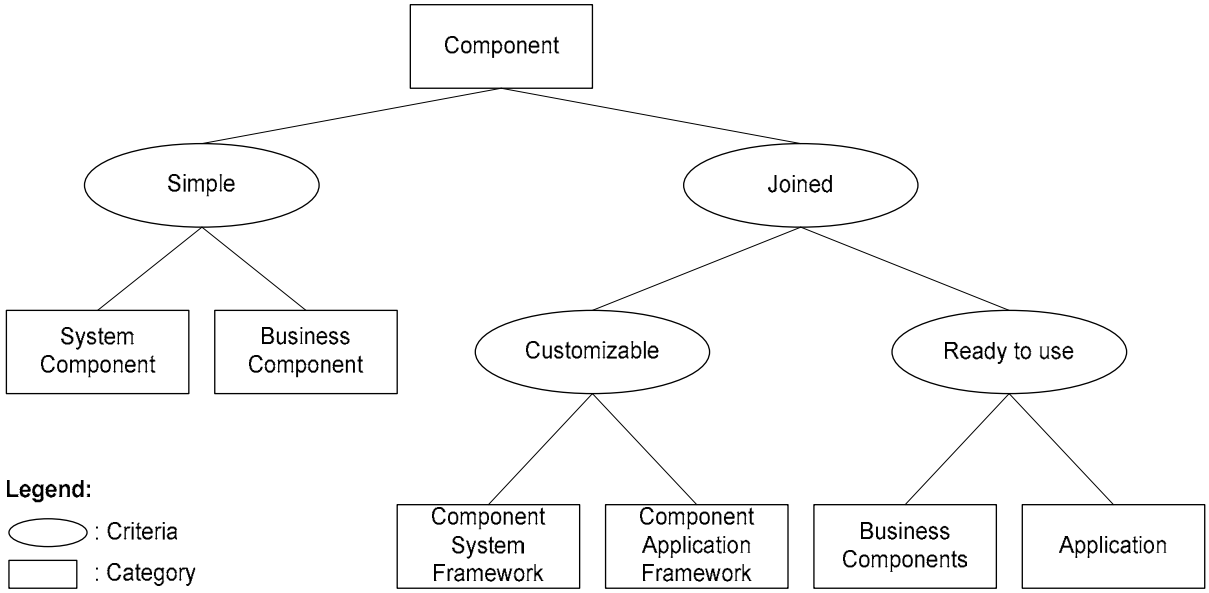


Fig. 1: Division into types of components

Firstly, components are categorized in simple (but self-contained) and joined components, as shown in Fig.1. Simple components can be subdivided in system components (serving generic functions) and business components (provide application specific functions). Joined components are subdivided in components which can be customized before using them in a specific context and those, which are ready to use, if necessary in a large number of variants.

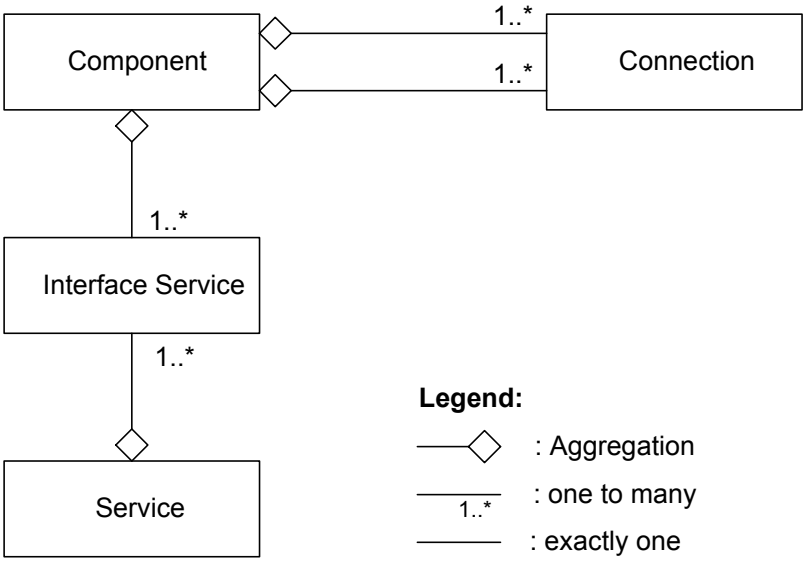


Fig. 2: Meta scheme of a component model

Joined and customizable components are called *Component System Framework,* when they are used to realize generic functions and *Component Application Framework,* when used in an appli-

cation specific context. One kind of customizing frameworks is their combination with simple or joined system and/or business components. Joined systems or business components are called Assemblies (an alternative term could be Configurations) in Fig.1. Applications which are ready-to-use configurations of components have to be differentiated from Assemblies.

The meta-scheme (Fig. 2), which is based on the (strongly simplified) component model, consists of a central object type COMPONENT. The bill of materials' like structure that describes its connection to the exterior is represented by the object type RELATIONSHIP. Towards its interior, the meta-schema describes the functionality and the tasks a component fulfills with the object types (standardized) SERVICE and INTERFACE SERVICE respectively.

As [OrRS1990] describes, besides the standardization of services it is also possible to standardize component attributes making use of data elements in the meta-scheme (Fig. 2). The essential aspect is that data elements as well as services for an application domain can be standardized independent of their purpose [OrRS1990].

2 Specification and Levels of Specification

In order to assemble business components with *little* effort to customer-individual application systems, it is necessary to establish *functional, content related* standards. In addition, a *methodical* standard has to be set, as intended with this memorandum. It provides the necessary framework, which notations have to be regarded for the specification of business components in order to simplify their reusability between companies and software developers. This is achieved by means of a notation mix that is standardized, accepted and well-known by all participating parties.

> *Specification* is defined as a complete, unequivocal and precise description of its external view that is which services a business component provides under which conditions.

User or author of a specification of a business component can be a variety of persons acting in different roles, e.g. functional architects, software architects, consultants, buyers, retail agents, project leaders or software developers who are taking very different roles [Sahm2000]. Examples for roles that are related to the specification of business components are:

- *Configurator:* Configuring purchased components.
- *Assembler:* Installing and distributing self developed or adjusted components in their target environment.
- *Quality Guard:* Testing components against their specification, checking content and formal correctness of the specification.
- *Component Administrator:* Filing components in a repository and administrating the different versions of the components and establishing classification systems.

As stated in [Turo1999] and [Turo2001], the specification of business components is to be carried out on different description levels (Fig. 3). For all of these levels a specific notation language has to be used. The possible notation languages (modeling and description languages) are:

- Mathematical or formal languages (algebra of sets, predicate logic, ...),
- Programming languages (imperative, functional, predicating and object oriented programming languages),

- Graphical languages or diagram languages (e.g. class diagrams, sequence diagrams, state diagrams),
- Normal or commonly used languages (e.g. every day language, technical language, standardized language).

In the following, the different description levels are going to be characterized and will be completed by a suggestion for a standardized specification method made by the authors of this memorandum. Detailed discussion of the various aspects and alternatives will be dealt within the chapters. Supplementary (graphical) notations take a special position, since they are meant to show *selected* facts additionally in a way that makes it easier to understand for certain readers.

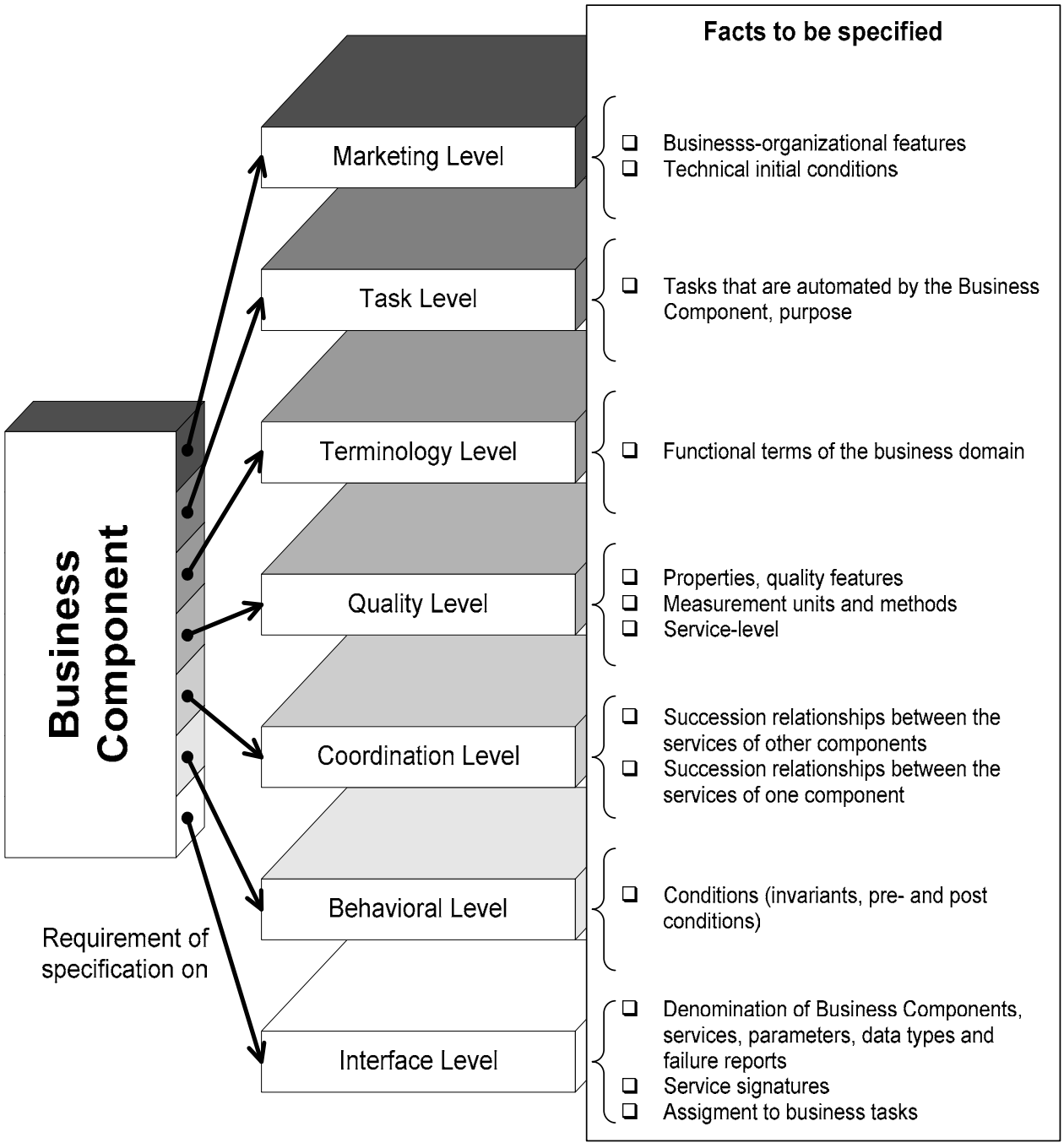


Fig. 3: Levels and facts to be specified

The *Primary,* standard specification method should be notations in formal languages, since they guarantee the necessary developer and company independent (inter-subjective) understanding of the specification results. A notation is *formal*, when syntax and semantics of the notation are precisely and unequivocally defined. The supplementary (*secondary*) specification method does not have to be a formal notation. In general, a lean notation mix should be established in order to facilitate a wide acceptance of specification methods for their good teachability and understandability.

It is to add, that some authors are skeptical concerning the use of formal specification methods for business components (or even parts of business components), since they consider the effort being too large and they fear a decrease of general understandability. As an example, the weaknesses of formal specification are demonstrated on the algebraic specification of abstract data types [cf. [BiMR1991, S. 288-291] and the references given there]. As possible alternatives for algebraic specification, there should be mentioned *operational* or *verbal* specifications. But it has to be underlined, that these specification results usually are criticized for their poor quality in terms of precision and inter-subjective understandability.

3 Interface Level

3.1 Purpose

On the Interface Level, basic technical arrangements are made. This means the denomination of services that are offered publicly by a business component, furthermore the public attributes, variables and constants, the definition of special (data) types (usually derived from basic data types), the definition of signatures of the offered services, and the declaration of error messages and exceptions. Additionally, the services required by a business component from other components have to be specified. As adequate form, e.g. programming languages or interface definition languages (*IDL*) can be used. The specified agreements guarantee, that service provider and service receiver can communicate. Above all, this layer is focusing on technical aspects of the communication, semantic aspects are remaining mostly disregarded.

Besides, the Interface Level is describing which (functional) terms that have been introduced on the Terminology Level, are related to which (data) types and which tasks introduced on the service level correspond to which services of the business component.

3.2 Notation Suggestion (Primary Notation)

It is suggested to use the Object Management Group’s (OMG) Interface Definition Language (IDL) [OMG2001a, S. 3.1-3.58]. The OMG IDL is well-suited to define the relevant facts that are to be specified on the Interface Level. It is an open, by science and industry accepted and commonly used standard. The operational semantics of the IDL, in particular the translation of language specific interface definitions, including object references and instances, to IDL, is described in the corresponding OMG *Language Mappings*.

To distinguish offered services from the required services from other components the latter are delimited by the key word `interface extern`.

For the notation of the corresponding (functional) terms and (data) types on the one hand side and tasks and corresponding services on the other hand two tables of two columns each are suggested.

3.3 Example (Primary Notation)

Fig. 4 shows a simplified excerpt from the specification for a Business Component “Stock Keeping” on the Interface Level. The component is meant to manage various stock accounts. The example of stock keeping has been chosen, since it is a business tasks which can be regarded as explored in detail and well understood [Kern1993, S. 63-74, 141-154], so the specification results can be easily understood from the specialized, business point of view.

```
interface StockKeeping {
        typedef string AccountNo;
        typedef double Quantity;

        struct Date { ... };

        struct Account {
                AccountNo       n;
                Quantity        SafetyQuantity;
                Quantity        ReorderingQuantity;
                ...
        };
        struct Booking {
                AccountNo       n;
                Date            ExecutionDate;
                string          OrderNo;
                double          BookingQuantity;
        };

        exception TooLittleQuantity {};

        void            Book(in Booking b);
        void            Reserve(in Booking b) raises (TooLittleQuantity);
        quantity        CalculateQuantityFor(in AccountNo n, in Date z);
        quantity        CalculateDemand(in AccountNo n, in Date z);
        void            Set SafetyQuantity(in AccountNo n,in double Quantity);
        void            Set ReorderingQuantity(in AccountNo n,in double Quantity);
};

interface extern {
        typedef long PeriodInDays;

        struct Date {
                unsigned short Day;
                unsigned short Month;
                unsigned short Year;
        };

        PeriodInDays CalculatePeriodInDays (in Date b, in Date e);
};
```

Fig. 4: Example for the use of OMG IDL on the Interface Level

By using the key word **`interface`**, the name of the component is determined. Subsections of the Business Component can herewith be identified unequivocally. `StockKeeping::Book` indicates, that a service `Book` belongs to the Business Component S*tock Keeping*. Subsequently, the simple and structured types are declared by using *predefined* types, which are necessary for the specification of interface signatures provided by this service.

After defining special 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 desired quantity cannot be reserved due to little (future) stock. Finally, (key word: **`interface`** `extern`) the services required by the Business Component are specified, as well as their calling parameters and return values.

Fig. 5 and Fig. 6 are showing examples of the corresponding functional terms and (data) types and the tasks and services respectively.

(Functional) Term	**`(Data) Type`**
account	`Account`
quantity	`Quantity`
booking	`Booking`

Fig. 5: Example for the notation of corresponding terms and types

Task	**`Service`**
Pass booking b	`void Book(in Booking b);`
Reserve booking b	`void Reserve(in Booking b) raises (TooLittleQuantity);`
Calculate quantity of account n for date z	`Quantity CalculateQuantityFor(in AccountNo n, in Date z);`

Fig. 6: Example for the notation of corresponding tasks and services

3.4 Outlook

As a possible future alternative to the Interface Definition Language, the use of the *Web Service Description Language* (WSDL) [CCM+2001] is being discussed.

4 Behavioral Level

4.1 Purpose

The specifications on the Behavioral Level serve as a detailed description of the Business Component's behavior. They are complementing the basic specifications of the Interface Level, which is describing above all the syntax of the interface, but leaving it open, how the component is behaving in general and especially in problem situations. E.g. it could be defined as invariant on the Behavioral Level, that the Business Component *Stock Keeping* requires a reordering quantity that is always greater than (or equal to) the safety quantity. Besides these invariants, on the Behavioral Level pre- and post-conditions for the services are specified.

The completeness of the specification on Behavioral Level is of great importance, which means not only specifying the provided services but the required services, as well. In this context the behavior has to be specified which is expected from services provided by other components.

4.2 Notation Suggestion (Primary Notation)

To specify facts on the Behavioral Level, the *Object Constraint Language* (OCL) is suggested. OCL has been accepted by the OMG as part of the Unified Modeling Language (UML) [OMG2001b] and subsequently is the recommended complement to the OMG IDL on the Behavioral Level.

To describe the services required from other components, an imaginary component “Extern” is defined, that provides these services.

4.3 Supplementary (Secondary) Notation

All conditions defined with OCL should be defined in natural language, as well.

4.4 Example

Fig. 7 shows the specification of the above mentioned stock keeping component on the Behavioral Level. Every condition in OCL will be supplemented by a specification in natural language (Fig. 8).

```
StockKeeping
        self.Account->forall(k:Account | k.SafetyQuantity >= 0)
        self.Account->forall(k:Account | k.ReorderingQuantity >= k.SafetyQuantity)

StockKeeping::CalculateQuantityFor(n:AccountNo,z:Date):Quantity
        pre:    self.Account->exists(k:Account | k.AccountNo = n)
        post:   result = self.Booking->iterate(b:Booking; r:Quantity = 0 |
                                if  b.AccountNo       =    n and
                                    b.Date           <=    z
                                then
                                    r + b.BookingQuantity
                                endif
                                )
```

Fig. 7: Example for the use of OCL on the Behavioral Level

Further examples can be found in the case study [Acke2001]. In the following, there will be additional explanation to the examples of Fig. 7.

The specification is beginning with a definition of the context to which the conditions belong. The context is emphasized by underlining. Hence, the first two specifications belong to the whole component `StockKeeping`. The specifications in the second paragraph are delimited to the service *`CalculateQuantityFor`* which is provided by the component. The delimitation of the context is shown by the **`::`** before mentioning the service. If parameters are needed to show the behavior of a component, they can be specified, as well. To specify the behavior of the service *`CalculateQuantityFor`* in the example, two typed parameters and the type of the return value are given. Conditions can occur as precondition (keyword **`pre`**), as post condition (key word **`post`**) or as invariant (no key word).

The example specifies two invariants. The first says that for every account (key word **`forall`**), that exists within the component `StockKeeping` a safety quantity has to be defined (`k.SafetyQuantity`) that is greater than zero. The key word **`self`** is signaling that the condition is referring to the delimited context (the entire Business component in the example), although it is restricted here by means of `.Account->` to a *collection* of accounts which exist within the component `StockKeeping`. The term *collection* means that the component contains a certain number of accounts, which can be accessed through the name `Account`, even though the im-

plementation of their management remains invisible to the exterior. For example, they could be organized in tables of a relational database, in a tree structure etc. *Collections* provide the possibility to specify complex functional details without having to refer to the actual realization.

The second invariant determines that a reordering quantity must be defined for all accounts, which is greater or equal the security quantity. Since these invariants have been specified in the context of `StockKeeping`, they apply to all services provided by this Business Component.

For the service *`CalculateQuantityFor`*, a precondition and a post condition have been specified: The precondition is stating, that the quantity can only be calculated for accounts which actually exist, that means, which have been set up (key word **`exists`**). The post condition describes, how the result of the service has been conceived (key word **`result`**). The collection of all bookings is searched (key word: **`iterate`**) for those that are referenced as `booking` (**`self`**`.Booking`) by the component. The start of this loop is set at zero (`r:Quantity = 0`). Within each iteration, it will be verified that the booking is referring to the correct account (`b.AccountNo = n`) and that the booking is executed after the point in time the parameter indicates that is handed over as deadline date (`b.Date <= z`). If these two conditions are fulfilled, the stock entry or exit is added to the final result (`r + b.BookingQuantity`).

The security quantity of every account has to be equal to zero or greater.
The reordering quantity of every account has to be equal to the security quantity or greater.
The service *`CalculateQuantityFor`* can only be executed by an account that is known to the component.
The quantity of an account (returned by the service *`CalculateQuantityFor)`* at a point of time is the sum of the quantities that have been booked on this account until this date. (Exits are considered as negative quantities and subtracted).

Fig. 8: Example for the use of secondary notation on the Behavioral Level

4.5 Alternative Notations and Outlook

In order to allow the use of components on the basis of specifications, the description on the Behavioral Level has to be extremely accurate. That is the reason for suggesting a formal notation. However, since a formal notation is very likely not to be understood by every reader, every condition should be complemented with an explanation in natural language. This will guarantee an enhanced understandability, especially for the communication with non-technical employees. The quality of these expressions in natural language can be enhanced by the use of sentence construction plans (see chapter 7/8)

5 Coordination Level

5.1 Purpose

The specifications on the Coordination Level are defining succession relationships between services and synchronization requirements. It can be stated for example, that security stock has to be defined for an account before the first booking can be carried out, or that two bookings cannot be executed on one account at the same point in time. Conditions on the Coordination Level can refer in the same manner to services offered by the Business Component as they can refer to required services from other components. If there is no need to stipulate exactly which component provides the required services, or the component is not known, the service will be addressed to the component "extern".

Purpose of the Coordination Level is providing relevant information how the Business Component can be integrated in a component based software solution from a process point of view. Hence, the specifications refer to conditions that have an economic, objective-logical relation to other components or to other services within the Business Component itself.

5.2 Notation Suggestion (Primary Notation)

For the specification of facts concerning the Coordination Level, it is suggested to use OCL with an addition of temporal operators [CoTu2000] as a notation standard. This choice allows avoiding a rupture in the use of notation methods between Behavioral and Coordination Level.

The following temporal operators can be used:

- *sometime_past* φ, *always_past* φ, φ *sometime_since_last* ψ, φ *always_since_last* ψ
- *sometime* φ, *always* φ, φ *until* ψ, φ *before* ψ
- *initially* φ

The expressions φ and ψ can hold a boolean value that is subject to changes in the course of time. E. g. φ could relate to the fact that in a given moment the quantity on the account is greater than or equal to the security quantity. The temporal expression "*always_past* φ" means that (from the perspective of the actual point in time) in the past φ has always been valid. "*Always_past* φ" is only true if in the past the quantity on the account has always been greater than or equal to the security quantity. For further details and exact semantics of the temporal operators, refer to [CoTu2000].

In addition, the operators *before* and *after* are introduced. The operand of both operators is a request of another service including calling parameters.

- The expression *before(service1(par1,par2))* is true in exactly that moment, when the service *service1* is being requested with the parameters *par1* and *par2*.
- The expression after*(service1(par1,par2))* is true in exactly that moment, when the execution of the *service1* with the parameters *par1*, *par2* has been successfully terminated.

Expressions such as *before(...)* and *after(...)* can be used as operands in conjunction with other temporal operators. By means of this extension, the request moment of methods can be expressed syntactically more precise with OCL expressions. Furthermore, it is possible to specify condi-

tions relating to different states during processing of a transaction. For further details and examples, refer to [Acke2001].

Moreover it has to be paid attention to the fact, that the use of temporal operators is restricting the use of OCL in the following aspects:

- An OCL expression cannot necessarily be interpreted at a specific point in time
- Temporal OCL expressions allow the use of Non-Query-Methods

As a consequence, temporal operators for specification purposes should be regarded as a modeling method for human-to-human communication. The use of temporal operators should be limited to the Coordination Level so that the Behavioral Level is not subject to the restrictions mentioned above.

5.3 Supplementary (Secondary) Notation

All specifications using the OCL extended by temporal operators should be complemented by explanations in natural language.

5.4 Example

Two examples of OCL extended by temporal operators are shown in Fig. 9. Every condition will be expressed in natural language, as well (Fig. 10).

```
OrderProcessing::PrintInvoice(at:Order)
pre : sometime_past( after(AcceptCustomerOrder(at) ) ) and
       not( after( CancelOrder(at) )
       sometime_since_last( after( AcceptCustomerOrder(at) ) ) )

OrderProcessing::AcceptCustomerOrder(at:Order)
post: sometime( after( PrintInvoice(at) ) ) or sometime( after( CancelOrder(at) ) )
```

Fig. 9: Example for the use of OCL extended by temporal operators

The first statement in Fig. 9 is a precondition for the service `PrintInvoice` in the Business Component `OrderProcessing`. This precondition expresses that before this service can be invoked, firstly the service `AcceptCustomerOrder` must have been executed for the same order and that secondly since that acceptance of this order no cancellation of this order has been carried out (by executing the service `CancelOrder`).

The second temporal condition is a post condition for the service `AcceptCustomerOrder`. In addition to the previous statement it is required that after accepting an order, sometime later either an invoice has to be printed for exactly this order by executing the service `PrintInvoice` or that this order must eventually be canceled by executing the service `CancelOrder`.

The two statements express different properties. On the one hand, the precondition for `PrintInvoice` does not forbid that the service `AcceptCustomerOrder` is executed without an invoice will ever being printed or a cancellation occurring for that order. On the other hand, the post condition for `AcceptCustomerOrder` does not exclude the execution of the service `PrintInvoice` for a certain order although this order has never been accepted by means of the service `AcceptCustomerOrder`.

An invoice for a customer order can only be printed, if the customer order has been accepted and it has not been canceled since accepting it for the last time.
For an accepted customer order in the future either an invoice has to be printed or it has to be canceled.

Fig. 10: Use of secondary notation on the Coordination Level

5.5 Alternative Notation Methods and Outlook

On the Coordination Level as well, conditions have to be specified with the highest degree of possible precision. For this reason, a formal notation should be used. In order to support an easy understandability, all formal specifications should be complemented by statements in natural language. For higher precision of the specification in natural language, sentence construction schemes can be used (see chapter 7/8).

6 Quality Level

6.1 Purpose

In addition to the mentioned levels until this point, which have been focusing on functional properties of a Business Component, the non-functional properties have to be specified as well. These are to be specified on the Quality Level. Examples are availability, performance properties or maintenance needs for the offered services. The specification on this level has to determine suitable quality criteria, the appropriate measures and methods for their actual measurement and, if appropriate, service level specifications for the services during runtime. Furthermore, it is to specify in which form this information is made accessible for the Business Component's user.

The measurement of a Business Component's quality properties can be realized by means of quantifying dynamic measurements such as throughput or response time. Certain Quality properties can also be inferred from static properties such as number and capacity of functions offered, linkage relations, or the completeness of test cases - with the corresponding empirical background experience.

In particular dynamic measurements depend to a significant extent on initial conditions (processor speed, main memory size, data base management system (DBMS), etc.) of the environment in which the Business Component is executed. These initial conditions have to be fixed to allow objective statements concerning quality properties.

However, the initial conditions might be very different for the various Business Components. The performance of one Business Component might depend primarily on the performance of the DBMS, where another is for the main part depending on processor velocity or network capacity. For this reason this notation standard leaves it up to the person who creates the specification to determine which initial conditions should be made definite for a special Business Component.

If a specification of initial conditions is possible, the determination of quality properties gives two choices:

- Specification of a reference environment: In the setting of a fixed reference environment that has been specified, quality results are measured. Nonetheless, this proceeding is rather costly and the specified reference environment can quickly be outdated and reduce the relevance of the measurements (concerning the intended use of the component).
- Collection of measurements in actual environments: For this proceeding, no detailed specification of a reference environment is defined. Instead, a variety of quality measurements is taken from a set of initial conditions, and these are collected within the specification. A problem for this proceeding might be the collection of data.

Regarding the mentioned initial conditions the finalization of quality criteria is not recommended, because this would limit the universality of the specification. For this reason, a rough framework is set to guide the process of determination of quality specifications which will be adapted by the manufacturer of the Business Component.

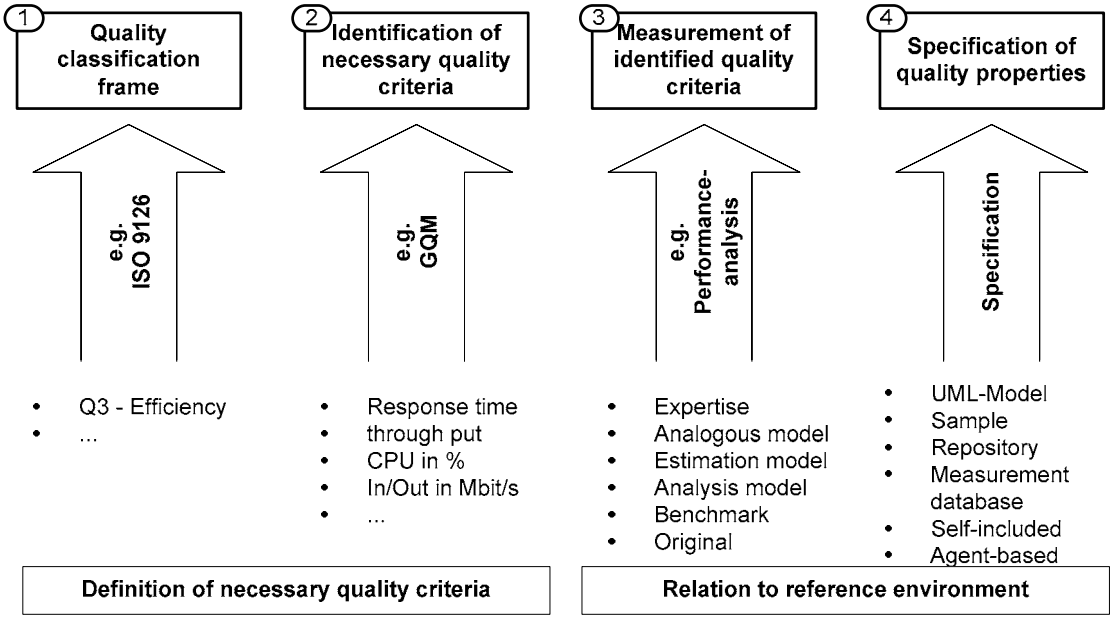


Fig. 11: Steps to quality specification [ScDu2001]

6.2 Steps for Quality Specification

Fig. 11 shows the necessary steps in the quality specification process, even though step 1 and 2 are independent of the aforementioned reference environment, where steps 3 and 4 can only be executed in the context of the reference environment. As an example, Fig. 11 shows the proceedings for a specification of the quality criteria “Efficiency”, the following sections cover a detailed description of the steps.

6.3 Quality Classes as Frame (Step 1)

The term “quality” for a software product in general and for Business Components can be interpreted in various ways. Therefore it is inevitable to specify a quality model, which makes the term “quality” operational. In order to meet this need various so-called *Factor Criteria Metrics*

(FCM) Model approaches have been developed (cf. [Balz1998]) which determine quality criteria on the basis of quality factors and suggest corresponding quantification metrics.

The ISO norm 9126 is a standard for quality models in accordance with the FCM approach. In the following, it will be used as an orientation guide to develop quality classes to evaluate components. Introducing these quality classes [ScSc2000] within the specification of software components allows the granular consideration of quality properties of a concrete component.

- *Portability Q1*: Adaptability, Installability
- *Usability Q2*: Learnability, Operability, Understandability
- *Efficiency Q3*: Spatial, in terms of time, in terms of resources
- *Functionality Q4*: Suitability, Interoperability, Precision
- *Reliability Q5*: Fault tolerance, Maturity, Recoverability
- *Maintainability Q6*: Analyzability, Changeability, Stability, Testability

A concrete Business Component e.g. can show the quality properties Q1 and Q3, which means that the specifications contain statements concerning exactly these properties, but not to the others. What the finalized instance of a certain quality class' specification looks like will be determined with the help of the Goal-Question-Metric (GQM) method.

6.4 Component Specific Quality Models (Step 2)

In order to identify the quality criteria that have to be specified for a business component, the so-called GQM Paradigm can be used [SoBe1999]. This offers a methodical proceeding model to work out the specific quality model tailored to the needs of a development process. Hence, it can be used for component development, as well. The GQM method is based on the quality objectives that have to be agreed upon (in the special case of quality, these can be taken from ISO 9126-1). Furthermore, by determining the questions how these objectives can be reached, the answer to these questions/the quantifications lead to the needed metrics. In order to answer the identified questions, success criteria are typically determined following [Dumk2001]:

- Point of View: Component developer, component user, buyer
- Deployment: Special project or special product class
- Purpose: Analysis, comprehension ...
- Context: e.g. in the setting of a development team.

Suggestions for external metrics can be found in ISO 9126-2, for internal metrics in ISO 9126-3. Furthermore, the GQM approach suggests other tasks to ensure an efficient metric measurement, result interpretation and validation, see [DuFS2000].

For the quality class Q3 (efficiency) the result can be as follows:

Goal

> The measured performance of the functions provided by the component is highly important for its successful use. (e.g. for components in sectors like telecommunication, banking, military)

Question

Which measurements are needed for the assessing of performance aspects related to time and space?

Metric

For the assessment of performance that is visible to the exterior, the measurement of response time and throughput of a concrete function is suggested (cf. ISO 14756).

In addition, it is necessary to determine, which resources have been used to achieve these results (CPU, bandwidth, software services etc.), the work load (in their temporal succession), the hard- and software architecture with the performance properties of the entire architecture (network, processors, services etc.), which is equivalent to the above mentioned reference environment.

For the quality class Q3, this proceeding for determining the measurements have been elaborated in [ScSc2000].

6.5 Measurement of the Determined Quality Criteria (Step 3)

Quantifying the quality criteria of a component can only be done by using concrete proceedings or methods, which should be tool based for efficiency reasons. In the following examples of these proceedings with respect to the quality classes are shown:

- *Portability Q1*: In the special case of an Enterprise Java Bean (EJB), tools (e.g. parser) could be used to analyze the conformity with EJB specifications.
- *Usability Q2*: Evaluation by using e.g. a question catalogue on the component's handling properties, which will be at the potential user's disposition.
- *Efficiency Q3*: Use of methods such as performance estimation, performance models, performance benchmark and the suitable profiler tools.
- *Functionality Q4*: Execution of functionality tests during the development phase.
- *Reliability Q5*: Extraction of experience by measuring deployed components in real world scenarios.
- *Maintainability Q6*: Software evaluation to gain static code metrics

Following the IEEE standard, the subsequent, basic measuring strategies can be used: evaluation (e.g. with questionnaires), feature estimation (e.g. with a formula based description of potential relations), model based measurement or direct measurement.

6.6 Specification of Quality Properties (Step 4)

Since there are too many possible notations, no Primary notation can be suggested. Therefore, only some suggestions are given, how extracted quality features can be used as a specification for a component.

- Use of elements or diagrams according to the UML notation
- Use of a formula based description
- Storage of quality properties in the Business Component itself
- Storage of quality properties in a repository.

7 Terminology Level

7.1 Purpose

In all different levels, the specification of business components uses technical terms, which have a domain specific functional meaning (semantic). Generally, these terms do not have an unequivocal meaning and/or definition and, hence, have to be specified to guarantee their unequivocal use.

The Terminology Level serves as central registry for all terms of a component and keeps all terms which are useful for the specification and their definitions in a dictionary. The terms that are defined on this level are used on all other levels (e.g. for the specification on the Marketing Level or Task Level). For the composition of the dictionary, all (important) terms applied in the specification will be listed with a definition. This supports the Business Component's self-containment, as a precondition to being marketable. Furthermore it should be possible to additionally refer to a standard if some of the terms are related to it. In this way, functional terms can be redefined by way of derogation from a standard.

For producers of a Business Component as well as for users deploying it in their system, a comprehensive administration of functional terms with respect to all components is of great importance in order to locate conflicts based on different definitions. This task could be provided e.g. by a component repository, which helps administrating the business components.

Functional terms and their definitions can be stored in a computer accessible way, e.g. by using a *Resource Description Framework* (RDF), an XML based language to store terminologies. However, it is suggested to use a description in natural language or any other easy to read language for the specification on the Terminology Level. That is, because usually functional experts have to choose the component on basis of the terminology it uses.

Besides the dictionary of technical terms, the Terminology Level allows to optionally specify an integrated scheme of attributes and services giving an overview of the attributes and services the Business Component is using.

Although such an integrated scheme is not belonging to the conceptual and domain specialized intention of this level and shows characteristics of a technical description, it is reasonable to specify the scheme on this level. Firstly, the different levels (e.g. Coordination, Behavioral and Interface Level) can make use of it. Since this level has the character of a central reference book, it is logical to store it here. Secondly, the denomination of attributes and services (in a best case scenario) is closely related to the functional terminology.

It is to be stated in addition, that such an integrated attribute and service scheme is not intended to substitute the specification of the Business Component's inside view, but rather aimed at supporting the specification process on other levels.

7.2 Notation Suggestion (Primary Notation)

In order to meet the aforementioned requirements, it is suggested to specify the terminology by using a so-called Standardized Business Language as primary notation (cf. [Ortn1997] and [Lehm1998]). It is characterized by a dictionary of unequivocally defined functional terms and by

using a rational grammar (reconstructed grammar of natural, colloquial language) to form the statements. A rational grammar consists of patterns and stencils to compose sentences.

The composition of a dictionary for the component storing the functional terminology is the primary goal of the specification on the Terminology Level. This is done by applying methods for the definition and the arrangement (organization) of all relevant functional terms.

Methods for the definition are:

- Explicit definitions of terms (avoiding circular definitions)
- Predicator rules, allowing to put different terms in relation
- Introduction of new words with positive examples and examples to prove the opposite (in order to solve the problem of a starting point for definitions)

Methods for the organization of terms are:

- Alphabetical order
- Short and long definitions
- Usage examples

As a graphical illustration of *conceptual* relations a complementing construction of an integrated attribute and service scheme a *UML class diagram* is recommended. It has to be noted that this does neither predetermine an object oriented implementation nor suggest a special inside view of a component.

7.3 Example (Primary Notation)

A notation making use of the above mentioned recommendations could look like the following example of the specification for the financial accounting terminology (Fig. 12).

ASSETS

- ...

BALANCE

- Short definition: BALANCE = $_{DF}$ the comparison of ASSETS and LIABILITIES of a company at a special date (CUTOFF DATE)
- Long definition: BALANCE= $_{DF}$ a BALANCE is together with the PROFIT AND LOSS ACCOUNT part of the ANNUAL ACCOUNT. A BALANCE is the comparison of ASSETS and LIABILITIES of a company at a special date (CUTOFF DATE). A BALANCE has to be based on the GENERALLY ACCEPTED ACCOUNTING PRINCIPLES. [...]
- Examples: COMMERCIAL BALANCE, TAX BALANCE
- Predicators: $x \in$ BALANCE => $x \in$ ANNUAL ACCOUNT
  $x \in$ BALANCE => $x \notin$ PROFIT AND LOSS ACCOUNT

PROFIT AND LOSS ACCOUNT:

- ...

Fig. 12: Example for the specification of terminology

A specification of an integrated attribute and service scheme will not be depicted in this place, since the modeling of components with UML classes is generally known.

7.4 Complementary (Secondary) Notation

For the composition of a dictionary, the use of a formal standardized language can be omitted and hence, the dictionary will be in the form of natural language.

In addition, the use of a technical language as complementary notation would allow the caption of whole systems of technical terms. As an example, the XML-based RDF (cf. [Roth2001]) should be mentioned as one of the constituting technologies for a *semantic web.*

In this specification suggestion, no preference for a secondary notation in the dictionary is given, the same applies for the integrated service scheme and the attribute scheme.

7.5 Example (Secondary Notation)

Examples for dictionaries in natural language can be found in the corresponding scientific literature. In relation with financial accounting, dictionaries contain the functional terms, which have been used in the above mentioned example.

A definition of systems of functional terms by using RDF can be found e.g. in [Roth2001].

8 Task Level

8.1 Purpose

For the development of component oriented business application systems, Business Components are specified in order to support or execute different business tasks. The documentation of these tasks that are being supported by a Business Component, and in case the decomposition in subtasks on a content-related (conceptual) layer, bears an important range of advantages.

Firstly, this leads - in combination with the related domain (which is described on the Marketing Level) - to the field of application for a Business Component. The specification on the Task Level gives the functional experts an instrument to determine the suitability of a component for the actual use in a given situation.

In addition, the specification of business tasks (and its functional decomposition into subtasks) is a precondition to a reasonable assignment of these tasks to the services the Business Component is offering through its interfaces. Such an assignment is done on the Interface Level with respect to the Task Level (see chapter 3)

In the task index of a component, all tasks supported should be listed explicitly and, if applicable, structured into subtasks. This is supportive of the Business Component’s self-containment, which is a precondition for its marketability. Additionally there should be a possibility to reference a standard, if some of the tasks are related to such a standard. In this way, business tasks can be redefined as derogation from a standard.

8.2 Notation Suggestion (Primary Notation)

The specification of (business) tasks that are being supported by a Business Component should be departing from the content-related, conceptual point of view. This makes it preferable to use a language that is easy to read for end users. In addition, the description should use a clear language in order to avoid misunderstandings in the course of the documentation.

The primary notation is hence suggested to make use of a reconstructed Business Language for the specification, which is close to the natural language.

Such a reconstructed functional language is characterized by being based on a dictionary with clear and unequivocal (defined) technical terms. Furthermore, its propositions are made with a reconstructed grammar with a determined set of sentence construction plans as schemes. The dictionary is to be compiled in the specification of the Terminology Level whereas the Task Level is giving details about the business tasks and their functional decomposition into subtasks.

The structure of reconstructed functional languages is discussed among others in [Ortn1997] and [Lehm1998] As an introduction, a few sentence construction plans will be presented to illustrate the specification of functional statements.

- Abstracting relation (equality, inheritance):
  An A is a B (or a C)
- Compositional relation (dependency)
  An A consists of / is part of a B
- Distribution relation
  An A has a B
- Operation relation
  An A does / sustains a B
- Change relation
  An A becomes a B
- Succession relation
  An A happens before / during / after a B
- Historical relation
  An A is terminated before a B

The indefinite article a/an underlines that the sentence construction plans are being used to express general statements from which schemes can be derived.

8.3 Example (Primary Notation)

By means of the sentence construction plans, statements on the business tasks that a Business Component supports can be specified. The example of Fig.13 shows specification of a business component “Balancing”, which is used in financial accounting; it is supposed that every functional term (terms in capital letters) has been defined on the Terminology Level.

A BALANCE is part of an ANNUAL ACCOUNT
A BALANCE is based on a LEGAL FOUNDATION
A PROFIT AND LOSS ACCOUNT is made before a BALANCE

A BALANCE consists of a BOOKING, a PROFIT RESULT and a DOCUMENTATION
...

Fig. 13: Example for the specification of business tasks

8.4 Supplementary (Secondary) Notation

As an alternative to a specification in a reconstructed functional language, the natural language can be used (as prose). When a developer is deciding to use such a notation he/she is implicitly making use of the sentence construction plans that his/her natural language is based on. In order to achieve a certain degree of simplicity, complex sentence structures should be avoided. For the secondary specification as well, functional terms should be defined (on the Terminology Level) and a general understanding should not be assumed.

Due to the considerable problems caused by computer-supported administration of such a notation, it should be refrained from using natural language.

8.5 Example (Secondary Notation)

The example of section 8.3 can be described very similar in secondary notation (Fig. 14). However, the statements are not following regular schemes so that a (computer) system is not able to determine automatically which type of statement it is dealing with (and e.g. convert it into a diagram language etc.)

A BALANCE is part of the annual accounts.
It has to follow a LEGAL FOUNDATION
Before, the PROFIT AND LOSS ACCOUNT has to be settled
A balance consists of the subtasks BOOKING, PROFIT RESULT and DOCUMENTATION
...

Fig. 14: Example for the specification of business tasks in secondary notation

9 Marketing Level

9.1 Purpose

The purpose of the Marketing Level is to specify the features of the Business Component that are important from a *business-organizational* point of view. The level includes all those characteristics, which serve as a working basis for every sales person, customer, assembler, and quality ensuring officer for Business Components. Additional features of the Business Component, that are relevant for users in other roles, e.g. on the Behavioral or Quality Level, will not be regarded on this level. The features that are important on this level can be categorized in three sections [Kauf2000]:

- Business semantics features: Features that describe business related and semantic properties of the Business Component.
- Technical features: Features that are technical conditions for using and running the Business Component.
- Additional features: Features that are neither related to business-semantic nor to technical features.

9.2 Notation suggestion (Primary Notation)

As primary notation, the form of a table is suggested. The different attributes will be explained subsequently and are suggested to be specified using the following conventions:

- Every table entry gives the name of the attribute. It is printed **bold.**
- If it is optional to specify an attribute, it is put in square brackets [].
  Example: [optional attribute]
- Attributes that can be specified repeatedly, are put into braces {}.
  Example: {repeatable attribute}

Name An alphanumerical marketing name
[Identification] An unambiguous label to identify the component
Version This feature gives the version as well as the release of a component
Branch of Economic Activity The specification of the business domain gives supplementary information in which economic sectors a component can be applied. Basis for the definition of the sectors is the International Standard Industrial Classification of All Economic Activities [UNSD1989]. It is possible to choose more than one, or all sectors, as well as to select “independent from sectors”.
Domain To allow a rough content-related classification of the services provided by a component, a functional domain is defined. The delimitation of the functional domain has been defined by [Mert200]. The specifications are confined to the uppermost level of the Model of Functions. Possible values are: Research and development; sales; procurement; inventory; production; delivery; after sales services; finance; accounting; human resource; facility management. It is possible to choose more than one, up to all functional sectors.
Scope of Supply As stated a Business Component consists of the definition, of various (software) artifacts. The purpose of specifying the scope of supply is to determine which artifacts the component comprises. This is an important source for various persons in the software development process to complete their assigned tasks completely and correctly. Concerning the component production, the scope of supply is needed by the quality ensuring officers to check whether the release of a component corresponds to the (software) artifacts specified, whether the mentioned (software) artifacts exist in the release or if (software) artifacts have been added to the release erroneously. In the sense of a use of a Business Component, the scope

of supply is the basis for the installation of a component. For the assembler, the scope of supply bears the information whether all mentioned (software) artifacts have been included in the package and what purpose and functionality every (software) artifact has.
Component technology The specification of the component technology that has been used and the underlying version standards for this component.
{Systems requirements} In this feature, the systems requirements for a component specify the platform it can run on and the required systems configuration. A system configuration includes processor architecture, memory size, hard disk space, operating system, version of the operating system, component application framework and component system framework (middleware, data base system etc.)
[Manufacturer] This feature allows to identify the manufacturer of a component unequivocally which is needed for getting into contact with the manufacturer, as well as to facilitate gathering information from other sources. Further information could be quotations, turnover and its development, profits, economic development, reputation, number of installations, reference customers as well as reports from other customers [Kauf2000]
[Contact person] This feature names a first contact person in the case that a potential buyer is interested in purchasing the component from the manufacturer.
[Contract conditions] This feature can be used to specify conditions for buying the component from the manufacturer. These conditions are prices (per license, per user etc.), service level, terms of business, prices for service contracts, possible consulting services and training, terms of payment, legal domicile, etc.
[Comments] This feature allows specifying further aspects of relevance concerning the component. These could be e.g. parallel processing of different clients with a single instance of the Business Component, multiple user handling, or alike.

9.3 Example (Primary Notation)

The following example is a specification of the business component “bank codes”.

Name Bank codes	
Version V 1.0	
Branch of Economic Activity Independent of domains	
Domain Finance	
Scope of Supply	
bankcodes.jar:	File of java classes for the implementation

bankcodes.tws:	Component IDL for middleware Twister by Brokat
create_db.sql:	SQL script to generate an Oracle data base storing the codes
blz0010pc.txt:	Original data source of bank codes of the Central Bank of the Federal Republic of Germany (Source: http://www.bundesbank.de/)
SATZ188.doc:	Description of the original data source blz0010pc.txt (Source: http://www.bundesbank.de/)
script_sampledata.pl:	Perl script to convert the files SATZ188.doc into SQL instructions
bankcodestests.jar:	Dummy client implementations to test the component functionality

Component technology

Brokat Twister 2.3.5

Systems requirements

Processor architecture:	x86
RAM:	512 MB
Hard disc:	10 MB
Operating system:	Windows NT 4.0, SP 3
Data base system:	Oracle 8i
Component System Framework:	Brokat Twister 2.3.5

Manufacturer

Technische Universität Chemnitz, Professur Wirtschaftsinformatik II, D-09107 Chemnitz

Contact person

Peter Fettke, +49/371/531-4362, peter.fettke@isym.tu-chemnitz.de

Contract conditions

There are no general contract conditions but they have to be negotiated individually. In principle, research institutions and research interested industry company are addressed.

9.4 Alternative Notations, Deciding Aspects, Outlook

For reasons of good readability the table form has been preferred to more formal notations as primary notation. For the realization of specification elements on this level, another approach [FeLo2001] is suggested additionally, that is based in the *Extensible Markup Language* (XML) and a suitably specified *Document Type Definition* (DTD). Additional features that are relevant to the Marketing Level, are listed in [Kauf2000, S. 110-115], [Ohle1998, S. 135-137] and [W3C1997]

Literature

[Acke2001] *Ackermann, J.*: Fallstudie zur Spezifikation von Fachkomponenten. In: *K. Turowski (Ed.)*: Modellierung und Spezifikation von Fachkomponenten: 2. Workshop. Bamberg 2001, S. 1-66.

[Balz1998] *Balzert, H.*: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag, Heidelberg 1998.

[BiMR1991] *Biethahn, J.; Mucksch, H.; Ruf, W.*: Ganzheitliches Informationsmanagement: Daten- und Entwicklungsmanagement. Bd. 2, Oldenbourg, München 1991.

[CCM+2001] *Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.* Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, Download on 2002-02-06.

[CoTu2001] *Conrad, S.; Turowski, K.*: Meeting Specifications Demands for Business Components. In: *Siau, K., Halpin, T (Ed.).*; Unified Modelling Language: Systems Analysis, Design and Development Issues , Idea Group Publishing, Hershey PA, USA, 2001 :

[Dumk2001] *Dumke, R.*: Software Engineering - Eine Einführung für Informatiker und Ingenieure: Systeme, Erfahrungen, Methoden, Tools. Vieweg-Verlag, Braunschweig 2001.

[DuFS2000] Dumke, R.; Foltin, E.; Schmietendorf, A. (2000): Metriken-Datenbanken in der Informationsverarbeitung. Preprint der Fakultät für Informatik Nr. 8, Otto-von-Guericke-Universität Magdeburg. Magdeburg.

[FeRT1999] *Fellner, K.; Rautenstrauch, C.; Turowski, K.*: Fachkomponenten zur Gestaltung betrieblicher Anwendungssysteme. In: IM Information Management & Consulting 14 (1999) 2, S. 25-34.

[FeLo2001] *Fettke, P.; Loos, P.*: Ein Vorschlag zur Spezifikation von Fachkomponenten auf der Administrations-Ebene. In: *K. Turowski (Ed.)*: Modellierung und Spezifikation von Fachkomponenten: 2. Workshop. Bamberg 2001, S. 95-104.

[Kauf2000] *Kaufmann, T.*: Entwurf eines Marktplatzes für heterogene Komponenten betrieblicher Anwendungssysteme. Berlin 2000.

[Kern1993] *Kernler, H.*: PPS der 3. Generation: Grundlagen, Methoden, Anregungen. Hüthig Buch Verlag, Heidelberg 1993.

[Lehm1998] *Lehmann, F. R.*: Normsprache. Das aktuelle Schlagwort. In: Informatik-Spektrum 21 (1998) 5, S. 360-367.

[McIl1968] *Mcllroy, M. D.*: Mass Produced Software Components. In: *P. Naur; B. Randell (Ed.)*: Software Engineering: Report on a Conference by the NATO Science Comittee. NATO Scientific Affairs Division, Brussels 1968, S. 138-150.

[Mert2000] *Mertens, P.*: Integrierte Informationsverarbeitung 1: Administrations- und Dispositionssysteme in der Industrie. 12. Aufl., Gabler, Wiesbaden 2000.

[Ohle1998] *Ohlendorf, T.*: Architektur betrieblicher Referenzmodellsysteme - Konzept und Spezifikation zur Gestaltung wiederverwendbarer Norm-Software-Bausteine für die Entwicklung betrieblicher Anwendungssysteme. Aachen 1998.

[OMG2001a] *OMG (Ed.)*: The Common Object Request Broker: Architecture and Specification: Version 2.5, September 2001. OMG, Framingham 2001a.

[OMG2001b] *OMG (Ed.)*: Unified Modeling Language Specification: Version 1.4, September 2001. OMG, Needham 2001b.

[Ortn1997] *Ortner, E.*: Methodenneutraler Fachentwurf: Zu den Grundlagen einer anwendungsorientierten Informatik. Teubner, Stuttgart 1997.

[OrRS1990] *Ortner, E.; Rößner, J.; Söllner, B.*: Entwicklung und Verwaltung standardisierter Datenelemente. In: Informatik-Spektrum 13 (1990) 1, S. 17-30.

[Roth2001] *Rothfuss, G.*: Content Management mit XML. Springer, Berlin 2001.

[Sahm2000] *Sahm, S.*: Organisationsmodelle zur komponentenorientierten Anwendungsentwicklung / terminologiebasierte Spezifikation von Fachkomponenten. In: *K. Turowski (Ed.)*: Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme, Siegen, Deutschland, 12. Oktober 2000, Tagungsband. Siegen 2000, S. 17-40.

[ScDu2001] *Schmietendorf, A.; Dumke, R.*: Spezifikation von Softwarekomponenten auf Qualitätsebene. In: *K. Turowski (Ed.)*: Modellierung und Spezifikation von Fachkomponenten: 2. Workshop. Bamberg 2001, S. 113-123.

[ScSc2000] *Schmietendorf, A.; Scholz, A.*: Spezifikation der Performance - Eigenschaften von Softwarekomponenten. In: *K. Turowski (Ed.)*: Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme, Siegen, Deutschland, 12. Oktober 2000, Tagungsband. Siegen 2000, S. 41-49.

[SoBe1999] *Solingen, v. R.; Berghout, E.*: The Goal/Question/Metric Method. McGraw Hill, 1999.

[Stat1993] *Statistisches Bundesamt (Ed.)*: Klassifikation der Wirtschaftszweige, Ausgabe 1993 (WZ 93). http://www.destatis.de/allg/d/klassif/wz93.htm, Download on 2002-02-15.

[Turo1999] *Turowski, K.*: Standardisierung von Fachkomponenten: Spezifikation und Objekte der Standardisierung. In: *A. Heinzl (Ed.)*: 3. Meistersingertreffen. Schloss Thurnau 1999.

[Turo2001] *Turowski, K.*: Spezifikation und Standardisierung von Fachkomponenten. In: Wirtschaftsinformatik 42 (2001) 3.

[UNSD1989] *United Nations Statistics Division (Ed.)*: International Standard Industrial Classification of All Economic Activities, Third Revision, (ISIC, Rev.3). http://esa.un.org/unsd/cr/family2.asp?Cl=2, Download on 2002-02-15.

[W3C1997] *World Wide Web Consortium (Ed.)*: The Open Software Description Format (OSD). http://www.w3.org/TR/NOTE-OSD.html, Download on 2001-04-25.

Authors :

Jörg Ackermann

Von-der-Tann-Str. 42, 69126 Heidelberg
E-Mail: joerg.ackermann.hd@t-online.de

Dr. Frank Brinkop

iteratec Gesellschaft für iterative Softwaretechnologien mbH
Inselkammerstr. 4, 82008 München-Unterhaching
Phone: +49(89)614551-0, Fax: -10
E-Mail: frank.brinkop@iteratec.de

Prof. Dr. Stefan Conrad

Ludwig-Maximilians-Universität München
Institut für Informatik
Oettingenstr. 67, 80538 München
E-Mail: conrad@informatik.uni-muenchen.de

Peter Fettke

Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Information Systems & Management
09107 Chemnitz
Phone: +49(371)531-4375, Fax: -4376
E-Mail: peter.fettke@isym.tu-chemnitz.de

Andreas Frick

ExperTeam AG
Niederlassung Dortmund
Emil-Figge-Straße 85, 44227 Dortmund
Phone: +49(231)9704-292/(-200), Fax: -299
E-Mail: andreas.frick@experteam.de

Dr. Elke Glistau

Otto-von-Guericke-Universität Magdeburg
Fakultät Maschinenbau
Universitätsplatz 2, 39106 Magdeburg
Phone: +49(391)671-2660
E-Mail: elke.glistau@mb.uni-magdeburg.de

Holger Jaekel

Oldenburger Forschungsinstitut für Informatik-Werkzeuge und -Systeme (OFFIS)
Escherweg 2, 26121 Oldenburg
Phone: +49(441)9722-125
E-Mail: holger.jaekel@offis.de

Otto Kotlar

Rohmer Str. 24, 60486 Frankfurt
Phone: +49(69)0795602
E-Mail: otto_kotlar@yahoo.de

Prof. Dr. Peter Loos

Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Information Systems & Management
09107 Chemnitz
Phone: +49(371)531-4375, Fax: -4376
E-Mail: loos@isym.tu-chemnitz.de

Prof. Dr. Heike Mrech

Fachhochschule Merseburg
Fachbereich Maschinenbau
Geusaer Str., 06217 Merseburg
Phone: +49(3461)46-3027
E-Mail: heike.mrech@mb.fh-merseburg.de

Prof. Dr. Erich Ortner

Technische Universität Darmstadt
Fachbereich Rechts- und Wirtschaftswissenschaften
Wirtschaftsinformatik I
Hochschulstr. 1, 64289 Darmstadt
Phone: +49(6151)16-4309; Fax: -4301
E-Mail: ortner@bwl.tu-darmstadt.de

Dr. Ulrich Raape

Fraunhofer Institut für Fabrikbetrieb und -automatisierung IFF
Sandtorstraße 22, 39106 Magdeburg
Phone: +49(391)4090-359; Fax: -93359
E-Mail: ulrich.raape@iff.fhg.de

Sven Overhage

Technische Universität Darmstadt
Fachbereich Rechts- und Wirtschaftswissenschaften
Wirtschaftsinformatik I
Hochschulstr. 1, 64289 Darmstadt
Phone: +49(6151)16-4309; Fax: -4301
E-Mail: overhage@bwl.tu-darmstadt.de

Stephan Sahm

itelligence AG
Westendstraße 16-22, 60325 Frankfurt am Main
E-Mail: stephan.sahm@itelligence.de

Dr. Andreas Schmietendorf

System- und Technologieentwicklung Bereich EP2
T-Systems Nova GmbH
Entwicklungszentrum Berlin
Wittestraße 30 N, 13509 Berlin
Phone: +49(30)43577-7531; Fax: -7507
E-Mail: andreas.schmietendorf@t-systems.com

Thorsten Teschke

Oldenburger Forschungsinstitut für Informatik-Werkzeuge und -Systeme (OFFIS)
Escherweg 2, 26121 Oldenburg
Phone: +49(441)9722-216
E-Mail: thorsten.teschke@offis.de

Prof. Dr. Klaus Turowski

Lehrstuhl für Betriebswirtschaftslehre, insbesondere Wirtschafts-informatik II
Universität Augsburg
Universitätsstraße 16, 86135 Augsburg
Phone: +49(821)598-4431; Fax : -4432
E-Mail: klaus.turowski@wiwi.uni-augsburg.de
URL: http://wi2.wiwi.uni-augsburg.de