Inf Syst E-Bus Manage (2013) 11:403–435
DOI 10.1007/s10257-012-0196-6

ORIGINAL ARTICLE

A method to support a reflective derivation of business components from conceptual models

Dominik Q. Birkmeier · Sven Overhage

Received: 18 October 2011 / Revised: 19 May 2012 / Accepted: 26 June 2012 /
Published online: 11 August 2012

Abstract To successfully compete in today's volatile business environments, enterprises need to consolidate, flexibly adapt, and extend their information systems (IS) with new functionality. Component-based development approaches can help solving these challenges as they support the structuring of IS landscapes into business components with a loosely coupled business functionality. However, the structuring process continues to pose research challenges and is not adequately supported yet. Current approaches to support the structuring process typically rely on procedures that cannot be customized to the designer's situational preferences. Furthermore, they do not allow the designer to identify and reflect emerging conflicts during the structuring. In this paper, we therefore propose a new method that introduces a rational, reflective procedure to systematically derive an optimized structuring according to situational preferences. Using a design science approach, we show (i) how the derivation of business components can be formulated as a customizable multi-criteria decision making problem and (ii) how conceptual models can be used to derive business components with an optimized functional scope. To evaluate the feasibility of the proposed method, we describe its application in a complex case that was taken from a German DAX-30 automobile manufacturer.

Keywords Component-based development · Information systems development · IS design · Business component identification · Design science

Preliminary findings of this research have been discussed in Albani et al. (2008) and Birkmeier and Overhage (2012).

D. Q. Birkmeier (✉) · S. Overhage
University of Augsburg, Universitaetsstr. 16, 86159 Augsburg, Germany
e-mail: dominik.birkmeier@wiwi.uni-augsburg.de

S. Overhage
e-mail: sven.overhage@wiwi.uni-augsburg.de

1 Introduction

The information systems (IS) strategy of enterprises is confronted with a variety of challenges today. On the one hand, volatile economic conditions and markets require enterprises to quickly adapt IS to changing business requirements and to add new functionality fast (McDonald et al. 2009). On the other hand, enterprises need to improve the cost-effectiveness of their IS landscapes by consolidating the implemented IS operations (McDonald et al. 2009). In this demanding context, modular component-based IS development approaches promise to bring many advantages (Herzum and Sims 2000; Vitharana et al. 2003). They help structuring IS into loosely coupled parts of business functionality such that the resulting IS landscape can be more easily adapted to changes or augmented with new functionality (Sharp and Ryan 2005). As individual parts—the so-called business components—can be reused in different applications, such a structuring can simultaneously help to consolidate the implemented IS operations (Sharp and Ryan 2005). For these reasons, component-based development rapidly attracted great attention both in research and industry.

The increasing significance of component-based IS development has led to the introduction of several methodologies which provide methods for the design, implementation, and assembly of business components (Brown 2000; Cheesman and Daniels 2001; Herzum and Sims 2000; Lim 1998; Szyperski et al. 2002). Nevertheless, the component-based development paradigm also continues to pose research challenges. Especially the question of how to structure the functional scope of IS into self-dependent, loosely coupled business components remains to be answered (Birkmeier and Overhage 2009). The way IS are partitioned into business components has a substantial impact on the adaptability of IS to changing requirements and on the reusability of the individual components, though (Gorton 2011). Generally, fine-grained business components which each implement a very specific functionality are easier to reuse in different application contexts than coarse-grained components with a complex functionality (Szyperski et al. 2002). A structuring into fine-grained components, therefore, helps consolidating the provided IS operations and avoiding redundancies. However, a fine-grained structuring complicates the assembly of new IS and their adaptation to changing requirements because the number of interfaces between the components and, hence, the communication between components increases (Szyperski et al. 2002). In each enterprise and according to its situational preferences, designers will therefore have to strive for a structuring that balances the required reusability of components with the aspired agility of the resulting IS.

Current methods that have been proposed to support the structuring of IS into business components do not adequately consider such situational preferences, however. Usually, they are based on fixed, implicit assumptions about the criteria that determine an optimal structuring. Differences in the Weltanschauungs of the method creator and the method user hence result in a way of using the method that differs from the creator's intentions (Omland 2009). To determine a structuring, many methods furthermore build upon automated procedures which can hardly be influenced or customized to the designer's preferences. Yet, as the structuring of IS into components is a creative process, designers rather benefit from so-called reflective methods “that help to identify and process the emerging conflicts” (Smolander and Rossi 2008). In this paper, we therefore propose a new method to structure IS into business components. Like other approaches, it supports the derivation of business components from conceptual models. However, the method proposed here introduces a rational, reflective procedure that allows designers to systematically determine an optimized structuring according to their design preferences and to evaluate the stability of the results afterwards. Building upon a design science approach (Hevner et al. 2004), we show *(i) how the derivation of business components can be formulated as a customizable multi-criteria decision making problem and (ii) how the business functions and information objects described in conceptual models can be systematically assigned to business components with an optimized functional scope*.

In the next section, we define basic concepts and discuss related work to motivate the research gap and link our work to that of others. After detailing the research method that our approach builds upon in Sect. 3, we introduce a procedure model to systematically derive business components from conceptual models in Sect. 4. We then present an implementation of our method as a tool and describe its application by taking a complex case from practice as example in Sect. 5. After positioning our method relative to the current state of the art and describing its limitations in Sect. 6, we conclude by discussing implications of our research and future directions.

2 State of the art and related work

2.1 Component-based IS development

Component-based development methodologies build upon a modular paradigm to structure the software part of IS into loosely coupled constituents, the so-called software components. A software component can broadly be characterized as “a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties” (Szyperski et al. 2002). To optimally fulfill these characteristics, each component needs to implement a self-contained functionality however. In the context of business information systems, the functionality of components is often determined from a conceptual, business-oriented rather than from a purely software-technical viewpoint. Accordingly, such components are called business components:

Definition 1 [*Business component* (Baster et al. 2001; Herzum and Sims 2000)] A business component is a self-contained package of functionality performing a specific business function within a technology framework. It is a software implementation of an autonomous business concept or business process.

The autonomy of components can be expressed in terms of their functional cohesion and their coupling with other components. An optimized IS structuring thereby should fulfill the principle of minimizing coupling (i.e. the interfaces to other components) and maximizing cohesion (Parnas 1972; Szyperski et al. 2002).

These are, however, two contradictory subgoals which need to be balanced (Hopkins 2000). Coupling and cohesion can be generally defined as follows, whereat concrete measures will be provided in Sect. 4.1:

Definition 2 [*Coupling and cohesion* (Vitharana et al. 2004; Wallnau et al. 2002)] The coupling of a business component is the extent to which elements within a component relate to other elements which are not in that component. The cohesion of a business component is the extent to which its elements are interrelated. The cohesion of a component can be measured by assessing the coupling of elements within that component.

At a first glance, the principle of minimizing coupling while maximizing cohesion seems to only reflect technical design considerations. Yet, several authors have emphasized how this principle also reflects general managerial goals in IS design (Li and Henry 1993; Chidamber et al. 1998; Vitharana et al. 2004; Gui and Scott 2008). Among them, Chidamber et al. (1998) provided a thorough analysis of relations between technical and managerial considerations for object-oriented systems. Vitharana et al. (2004) further extended these examinations towards a strategy-based design of business components. Figure 1 summarizes the stated effects of coupling and cohesion on the following managerial goals.

Cost effectiveness refers to the possibility to reduce expenditure during the development of an IS, while maintaining the desired outcome (Vitharana et al. 2004). In an empirical analysis of object-oriented software development projects, Chidamber et al. (1998) have identified that a higher level of coupling between parts of a system can be associated with lower productivity and greater design effort. This can be attributed to the necessarily increased amount of interactions between different system parts and the required larger and more complex interfaces. Hence, strong coupling is negatively correlated with cost effectiveness. A cohesive design, on the other hand, was found to have a positive effect on the cost effectiveness of a project. This can be explained with the fact that related elements are likely to be implemented in one system part, which again reduces the required amount of interactions between system parts.

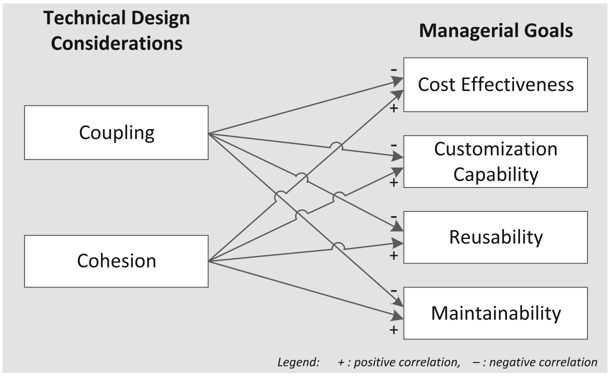


Fig. 1 Effects of technical features on managerial goals of IS development

Customization capability reflects the ease with which an IS can be customized through the combination of relatively independent system parts according to specific requirements. Vitharana et al. (2004) claim that the possibilities for customizing an IS are lowered with an increase in coupling, as a strong coupling makes the system parts more dependent on each other. They further argue that a strong cohesion within system parts, on the other hand, has a positive impact on the customization capability. This is based on the fact that functionalities realized in a highly cohesive component are more likely to be closely related towards each other than the functionalities provided in a less cohesive component of equal size. The requirements of an IS, however, are commonly interrelated, as well. Thus, if a cohesive component covers a certain requirement of an IS, it is likely that other related requirements are covered by the same component. Overall, the likelihood of a cohesive component to cover a whole subset of system requirements is, therefore, higher than for a less cohesive component of equal size. A survey performed by Vitharana et al. (2004) supports the claimed relationships.

Reusability refers to the extent of possible scenarios in which a part of a system can be seamlessly integrated in different IS development projects. For the object-oriented paradigm, Chidamber and Kemerer (1994) have shown that a tight coupling between objects hampers the reuse of system parts due to the persisting interdependencies. Equivalently, Vitharana et al. (2004) argue that the reusability of a single component is reduced if the strength of its dependencies towards other system parts increases. Hence, highly coupled components usually will need to be used together. A strong cohesion of system parts, on the other hand, has a positive influence on their reusability. Similar to the argumentation regarding the customization capability, a highly cohesive component is likely to satisfy related requirements more often than a less cohesive component, which increases the number of possible reuse scenarios (Vitharana et al. 2004). These correlations have also been empirically confirmed in three case studies (Gui and Scott 2008).

Maintainability relates to the effort which is necessary to implement requirement changes in an IS. Chidamber and Kemerer (1994) have shown that a strong coupling between system parts reduces the maintainability of object-oriented systems. Similarly, the maintainability of a component-based IS is negatively influenced by strong coupling between components, as fixes or enhancements of one component are likely to affect others as well and, thus, raise maintenance costs (Vitharana et al. 2004). A highly cohesive component is also presumed to be harder to maintain because of the many intra-component relationships (Vitharana et al. 2004). However, in view of the whole system, changes to a single, highly cohesive component are easier to maintain than changes affecting several interfaces and less cohesive components. Empirical studies of object-oriented projects have shown that coupling and cohesion measures are able to predict the maintainability of system parts during systems design (Li and Henry 1993).

A "good" component-based IS design (i.e. with cost-effective, customizable, reusable components and low maintenance costs) should hence aim at balancing the two conflicting structuring goals: to minimize the coupling between components, business functions and information objects should generally be grouped into a single business component whenever they are processed together at some point at run-time.

To maximize cohesion, a business component should only contain business functions and data which are all processed together at some point at run-time. As these design goals can be difficult to balance especially when structuring large IS with many components, methods that provide a systematic procedure for the derivation of business components are required to better support the structuring process. In line with the literature, we define the term method as follows:

Definition 3 [*Method* (Iivari et al. 1998; Fitzgerald et al. 2002)] A method is defined as a coherent and systematic approach, based on a particular philosophy of systems development, which will guide developers on what steps to take during the development of an information system. To that end, a method consists of a well-defined sequence of elementary operations which permits the achievement of certain outcomes if executed correctly.

According to this definition, a supporting method has to define the Weltanschauung of its inventor, i.e. the implemented strategy to structure IS into business components with certain properties. Ideally, it moreover ought to be reflective, which means that it can be customized according to project-specific design preferences and helps the designer to process conflicts that might emerge during the structuring process (Smolander and Rossi 2008).

2.2 Approaches to derive business components

From an architectural perspective, the software part of IS is constituted by operations, which represent the loci of computation, and data elements, which represent the processed information items (Shaw and Garlan 1996; Bass et al. 1998). Large software systems require decompositional mechanisms to make their design tractable (Shaw and Garlan 1996). To achieve this goal, different development paradigms along with specific methodologies that support their application in practice (Iivari et al. 1998) have been proposed. Among them, especially structured (module-based) development, object-oriented development, and component-based development have become widespread paradigms in practice (Szyperski et al. 2002). Each development paradigm introduces a specific strategy to structure IS into parts, however. As a consequence, approaches that support a systematic structuring of IS into modular parts vary according to the paradigm they build upon.

The *structured development paradigm* originally proposes to conduct a step-wise functional decomposition in order to disassemble systems into individual functions, which form the elementary parts of a system (Wirth 1971). As the functions resulting from such a decomposition are usually dependant on other functions that they interact with, module-based development approaches have been suggested to group interrelated functions into modules, which hide their internal implementation and form more self-dependant system parts than individual functions (Parnas 1972; Yourdon and Constantine 1979). While structured approaches hence organize IS according to their functional structure, the *object-oriented development paradigm* concentrates on the processed data types to disassemble a system into elementary parts, the so-called classes (Meyer 1997). In object-oriented designs, functions are considered to be secondary system elements which are assigned to the data types they operate upon (Meyer 1997). Compared to modules, which can encompass an arbitrary number of data types, classes are meant to be more fine-grained as they should each encapsulate only a single data type (Meyer 1997; Iivari 1995). In addition, they are more tightly coupled than modules since additional interdependencies such as inheritance relationships are introduced in object-oriented system designs (Szyperski et al. 2002).

In contrast to the before-mentioned development paradigms, which respectively focus on a single system constituent as structuring criterion, the *component-based development paradigm* follows a more general strategy to structure IS into parts. It aims at identifying self-dependant system parts of arbitrary size, i.e. components which should have minimal relationships to other components (Szyperski et al. 2002). To identify such system parts, the designer has to take into account all probable causes for interrelationships between system constituents. Between components, such interrelationships have to be kept at a minimum by grouping strongly interrelated, cohesive system constituents into a single component and by assigning loosely related system constituents to different components (Szyperski et al. 2002). Thereby, designers have to examine both system functions and data elements for connections that might exist between them.

As the component-based structuring strategy significantly differs from both the module-based and the object-oriented way of structuring systems, several specific approaches to support the derivation of software components in general and business components in particular have been proposed in literature. These approaches vary significantly with respect to their central characteristics, though. Building upon and updating a recent survey conducted by Birkmeier and Overhage (2009), we therefore examine approaches for the derivation of components with respect to their underlying Weltanschauung and their support for a reflective structuring process (Table 1).[1]

Noteworthy is the fact that a number of approaches does not even provide a statement about the component definition underlying the structuring technique (Kang et al. 1998; Lee et al. 1999; Sugumaran et al. 1999; Lee and Xiong 2001; Blois et al. 2005; Albani et al. 2008). Without any knowledge about the employed component definition, it is difficult to assess the suitability of these approaches at all, though. Among the other approaches, only very few explicitly build upon business domain oriented, conceptual component understandings like the one given in Definition 1 (Levi and Arsanjani 2002; Meng et al. 2005; Cai et al. 2011). While a business domain oriented understanding is focusing on the business functionality that is provided by software components, other understandings emphasize architectural or technical aspects. Approaches building upon such understandings are not straightforwardly suited to properly structure IS into business components

[1] Note that we will not explicitly discuss approaches to structure IS into software services although service-oriented development can be regarded as a special form of component-based development (Szyperski et al. 2002). While we have examined several approaches to identify services during our analysis of related work, we found that they were generally less mature than those to derive software components. Our findings are in line with recent surveys which also provide explanations for this observation (Birkmeier et al. 2009; Kohlborn et al. 2009).

Table 1 Overview of related work (based on Birkmeier and Overhage 2009)

	IBM Corporation (1984)	Kang et al. (1998)	Lee et al. (1999)	Sugumaran et al. (1999)	Cheesman and Daniels (2001)	Ganesan and Sengupta (2009)	Jain et al. (2001)	Lee and Xiong (2001)
Component definition	–	None	None	None	Architectural	Architectural	Architectural	None
Degree of formalization	Structured	Recommendations	Structured	Recommendations	Structured	Structured	Algorithm	Algorithm
Direction	Top-down	Meet-in-the-middle	Top-down	Meet-in-the-middle	Top-down	Top-down	Top-down	Top-down
Identification strategy	Matrix analysis	Other	Matrix analysis	Other	Other	Matrix analysis	Clustering analysis	Graph-based
Optimizing approach	■	□	■	□	□	□	■	□
Model views	Processes and data	–	Processes, data and functions	Processes	Processes and data	Processes and data	Processes and functions	Processes and functions
Existing structures and system dependencies	□	□	□	□	□	□	□	□
Tool support	□	□	□	■	□	□	□	□
Quality assertions	None	None	Evaluation	None	None	None	Evaluation	None

Table 1 continued

	Levi and Arsanjani (2002)	Jang et al. (2003)	Kim and Chang (2004)	Blois et al. (2005)	Meng et al. (2005)	Rodrigues and Barbosa (2006)	Albani et al. (2008)	Cai et al. (2011)	Cui and Chae (2011)
Component definition	Domain-oriented	Technical	Architectural	None	Domain-oriented	Technical	None	Domain-oriented	Technical
Degree of formalization	Recommendations	Structured	Structured	Structured	Algorithm	Algorithm	Algorithm	Algorithm	Recommendations
Direction	Top-down	Meet-in-the-middle	Meet-in-the-middle	Bottom-up	Top-down	Bottom-up	Top-down	Top-down	Bottom-up
Identification strategy	Other	Matrix analysis	Matrix analysis	Other	Clustering analysis based on graph	Graph-based	Graph-based	Clustering analysis based on graph	Clustering analysis
Optimizing approach	□	■	■	□	■	■	■	■	■
Model views	Processes	Processes and functions	Processes, data and functions	–	Processes, data and functions	Functions	Processes and data	Functions and data	Functions
Existing structures and system dependencies	□	□	□	□	□	□	■	□	□
Tool support	□	□	□	■	□	■	■	■	■
Quality assertions	None	None	Evaluation	None	Evaluation	None	Evaluation	Evaluation	–

with an autonomous business functionality, however. Some of them, for instance, propose to analyze source-code and database schemes to derive software components as higher-order design abstractions (Jang et al. 2003; Rodrigues and Barbosa 2006). Accordingly, the business functionality of the resulting components is not of an immediate concern.

Approaches for the derivation of components furthermore differ in their support for a systematic procedure. Some do not fulfill the criteria of a method as given in Definition 3 and only provide general recommendations that should be considered during the structuring process (Kang et al. 1998; Sugumaran et al. 1999; Levi and Arsanjani 2002; Cui and Chae 2011). While such recommendations may serve as criteria to analyze the quality of a structuring, they provide little guidance for the actual process. Other approaches, therefore, introduce concrete steps to achieve a structuring into components. A few of these approaches define taxonomies of component types, e.g to structure an IS into process components and data components (Cheesman and Daniels 2001; Blois et al. 2005). While IS can be structured into such component types straightforwardly, the resulting components are not optimized with respect to the design principle of minimizing coupling and maximizing cohesion.

To achieve results which are better suited to fulfill this design principle, a group of so-called optimizing approaches introduces more flexible ways of structuring IS into components. Among these approaches, many utilize so-called matrix analyses to derive components from business domain or IT models (Lee et al. 1999; Jang et al. 2003; Kim and Chang 2004). A matrix analysis was first used in the Business Systems Planning method (IBM Corporation 1984). This method describes how business information systems can be systematically decomposed into packages based on the functionality to be provided and data to be processed as specified in business process models. Although this approach is not based on the structuring strategy of maximizing self-dependence in general, it has also been recommended to derive business components from conceptual models in the past (Albani et al. 2003). To achieve this goal, the individual business functions and the processed business data are denoted as two dimensions of a matrix. Thereafter, relationships between business functions and data (such as create or usage relationships) are charted in the matrix. They serve as the basis to cluster business functions and data according to the design principle of minimizing coupling and maximizing cohesion. Methods that only build upon analyses of two-dimensional matrices are limited regarding the information that is used to derive self-dependant business components, though. Focusing solely on relationships between business functions and data can compromise the design results as other aspects like dependencies between functions or between data elements are ignored (Birkmeier and Overhage 2009).

To take additional kinds of relationships into account during the optimization, clustering analyses as well as graph-based methods have been proposed to identify business components (Jain et al., 2001; Meng et al. 2005; Albani et al. 2008; Cai et al. 2011). While such techniques are basically able to work with arbitrary kinds of relationships, the design of the proposed methods is restricted to cover pre-defined sets of relationships only (Birkmeier and Overhage 2009). Designers are hence not able to customize the proposed methods according to the requirements of their projects and to the information that is available. Furthermore, the fact that most projects are not part of greenfield IS developments but rather have to take legacy structures in the form of already existing components into account, is commonly left aside (Birkmeier and Overhage 2009). Only one approach provides some initial support to take legacy components into account during the structuring process (Albani et al. 2008).

A major disadvantage of both—the optimizing approaches and the other structured approaches—is, however, that they either specify the derivation of components as work steps which have to be performed manually (IBM Corporation 1984; Lee et al. 1999; Jang et al. 2003; Kim and Chang 2004) or introduce completely automated derivation procedures which cannot be influenced by the designer at all (Jain et al. 2001; Meng et al. 2005; Albani et al. 2008; Cai et al. 2011). Especially in industry-sized IS development projects, manually deriving components based on matrix or clustering analyses burdens the designers with significant cognitive load, though. It then strongly "depends on the knowledge of the designer if [...] useful results can be achieved" (Birkmeier and Overhage 2009). Completely automated methods, on the other hand, do not allow for a sufficient level of influence and control during the derivation process. Because IS design is a significantly creative process, "it is important that the individual developers engage their competencies" (Fitzgerald et al. 2002; Omland 2009).

Although the design principle of minimizing coupling and maximizing cohesion has been taken into consideration by a variety of approaches, there hence remain several shortcomings. First of all, a method to support the structuring of IS has to make explicit the Weltanschauung of its creator. As many of the optimizing approaches do not even provide details about the component definition that underlies the structuring procedure, it remains difficult to assess their suitability for concrete scenarios. Worse is the fact that these approaches cannot be customized to project-specific preferences, because the design of the proposed approaches is restricted to cover only a fixed set of structuring determinants and/or legacy components cannot be taken into account. Completely automated structuring procedures, furthermore, hamper the chance to achieve outstanding solutions in IS design, as the designer's expertise and creativity are suppressed. To address these shortcomings, we propose an optimizing method for the derivation of business components that can be customized and involves the designer's expertise throughout the structuring process. While the core derivation step itself is automated to a considerable part in order to ensure an optimized design, the method provides several options which support its customization to situational project preferences. It, moreover, contains a number of steps to evaluate and reflect upon both the structuring preferences as well as the identification results.

3 Research method

The development of the reflective method to derive business components from conceptual models follows the *design science paradigm* (Hevner et al. 2004). This paradigm aims at a rigorous construction of innovative IT artifacts which can be constructs, models, methods, and instantiations (March and Smith 1995). Most importantly, it thereby is the novel contribution to the existing knowledge base that distinguishes design science research from the practice of building IT artifacts (Hevner et al. 2004). Design science research generally is meant to solve a relevant problem by developing a novel artifact (March and Smith 1995). Accordingly, the artifact has to be scientifically evaluated in order to prove that the research goals driving the artifact construction indeed have been fulfilled (Iivari 2007). To show that the reflective method presented in this paper does bring advantages for the designers during the derivation of business components, we evaluated it in three steps that we will describe in detail later on: (i) to emphasize the *contribution* to the existing knowledge base, we position the presented method relative to the existing approaches described in Sect. 2; (ii) to demonstrate the *feasibility* of the method, we describe its instantiation as part of a software tool; (iii) to illustrate its *usefulness*, we describe and discuss its practical application in a complex system design task.

Moreover, design science research can be distinguished from the practice of designing IT artifacts by the rigor with which the artifact is constructed (Iivari 2007). The construction, hence, should follow a rigorous, scientific process. For the development of the presented method, we followed the *design cycle* that was introduced by Takeda et al. (1990) and augmented for the use in design science research projects by Vaishnavi and Kuechler (2004). The augmented design cycle defines the problem statement, solution concept, solution instantiation, and evaluation as milestones of design science research projects. To make transparent how we followed this construction process, we provide details about each stage of the augmented design cycle in the paper. While we have already motivated the research gap and presented the problem statement in the last sections, we will describe the solution concept, its instantiation, and the conducted evaluation in the remainder of this paper.

4 Business component derivation

4.1 Procedure model

The presented method supports designers during the derivation of an initial component structure. As a systems development method, it is embedded into an overall component-based development process. In parallel with the formation of component-based methodologies, various specific development process models have been proposed in literature (D'Souza and Wills 1999; Brown 2000; Cheesman and Daniels 2001). These component-based development process models share many similarities. In practice, especially the comparatively lightweight process model defined by Cheesman and Daniels (2001) has become widespread. It describes a development workflow that includes the phases Requirements, Specification, Provisioning, and Assembly, followed by Test and Deployment (Fig. 2).

During the initial *Requirements Definition* phase, all relevant business requirements are described and summarized together with the conceptual models that describe the required details of the business domain. The *Specification* phase itself is divided into three sub-phases, namely Component Identification, Component

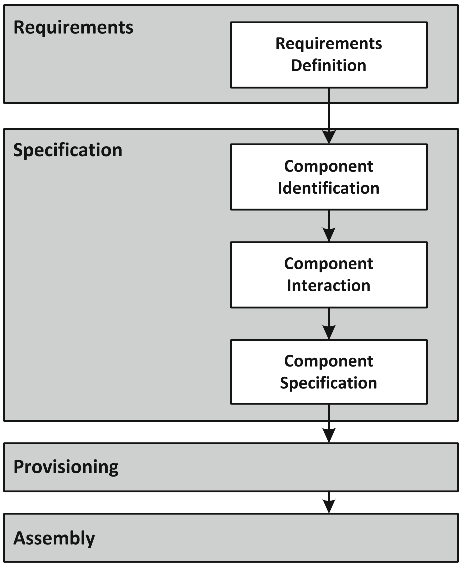


Fig. 2 Overall development process model for component-based systems (Cheesman and Daniels 2001)

Interaction and Component Specification. During the *Component Identification* stage, the previously defined business requirements and the provided conceptual models are used "to identify [...] an initial component architecture" as system structuring (Cheesman and Daniels 2001). The initial system structuring is driven by the description of the business domain and is done independently from technical considerations. The derived component structure builds, however, the basis for all successive technical specifications. During the *Component Interaction* phase, the system structuring is refined from a technical perspective. In this stage, the detailed design is done which covers all aspects "down to the individual operation level" (Cheesman and Daniels 2001). Especially, the operations and interfaces between the components have to be determined. The final stage of *Component Specification* is approached once the architecture is stable. It results in a detailed technical specification of operations, states of component objects, pre- and postconditions, as well as other constraints. The purpose of the remaining phases is the implementation of the specified components or their selection from a repository of existing ones (*Provisioning*), and the *Assembly* of the component architecture.

With respect to the described development process model, the presented method supports the *Component Identification* phase. To deliver an initial component structure, it evaluates the information about the business domain that is contained in the conceptual models which are used as input.

Definition 4 [*Conceptual model* (D'Souza and Wills 1999; Topi and Ramesh 2002; Hadar and Soffer 2006)] A conceptual model—sometimes called business or domain model—describes the users' world in a modeling language independently of implementation technology and constraints. It explains what concepts there are, how they are related, and how they interact. Conceptual modeling is applied in the early phases of information systems analysis and design. Amongst others, it commonly includes information about the business processes, the business information objects, and the composition of business functions.

Per se this definition is not restricted to "informational" models, but also includes models that, amongst others, describe the flow of material resources in a business process. As our approach focuses on the structuring of information systems, however, material resources are not considered in the analysis. Generally, it is important that methods utilize as most detailed information about the required business functionality as possible. The available information, however, is typically distributed over various models instead of being contained in a single conceptual model. These models can be available in various modeling notations—e.g. Business Process Model and Notation diagrams for business processes, Entity Relationship Diagrams for information objects, and Function Decomposition Diagrams for business functions.

The introduced method uses this information and takes a top-down, business-oriented view. From a system-theoretic perspective, the approach must ensure that all components together exhibit the behavior expected from the whole system. Therefore, set-theoretic considerations are used to formalize the derivation of business components. In such a derivation scenario, a set of elements from conceptual models is considered together with a set of relationships between those elements. Depending on the specific input models, the extracted elements in the set might typically be business process steps, information objects, and actors. Consequently, characteristic relationships are, amongst others, control flows between process steps or entity relationships between information objects. In Sect. 4.2, we discuss the usage of the most common elements and models for our approach. However, depending on the specific project, the designer generally ought to be able to include further models, elements, and relationships into the derivation process. Additionally, relationships between elements could be of different types and importance. To reflect the latter, a designer further ought to be able to assign a weight $w_{(u,v)}$ to a relationship between two elements u and v.

To derive business components, the set of elements needs to be decomposed into a partitioning $P = \{C_1, C_2, \ldots, C_k\}$ of pairwise disjoint subsets C_i, such that the union of all components $\bigcup_{i=1}^{k} C_i$ matches the whole set. Thereby, k resembles the total number of components. In this context, coupling and cohesion can be formalized into concrete measures.

Definition 5 [*Coupling and cohesion measures* (Vitharana et al. 2004)] Measures of coupling (1) and cohesion (2) for a set of business components P are defined as:

$$Cou(P) = \frac{1}{2}\sum_{i=1}^{k}\sum_{u \in C_i}\sum_{v \notin C_i} w_{(u,v)} \tag{1}$$

$$Coh(P) = \frac{1}{2}\sum_{i=1}^{k}\sum_{u \in C_i}\sum_{v \in C_i, u \neq v} w_{(u,v)} \tag{2}$$

Therefore, the minimizing coupling—maximizing cohesion principle depends on the weights of relationships running between and within business components. An optimized solution, thus, has to respect the individual weights. Higher weights imply a more important relationship between two elements and, therefore, separating those elements into different business components constitutes a stronger impact onto the overall partitioning. Following the objective of minimizing coupling in a component structure, the target function of the optimization problem, hence, can be stated as: *Minimize the sum of weights of all relationships between elements in different business components* (3, left). This target is, furthermore, mathematically equivalent to the counterpart task of *maximizing the sum of weights of all relationships between elements in the same business component* (3, right). The second goal resembles the maximizing cohesion principle.

$$\min_P \frac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\notin C_i} w_{(u,v)} \Leftrightarrow \max_P \frac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\in C_i,u\neq v} w_{(u,v)} \tag{3}$$

Proof The equivalency follows from the fact that the sum of weights of all existing relationships (4) remains constant:

$$\sum_{\forall u,v} w_{(u,v)} = \frac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\notin C_i} w_{(u,v)} + \frac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\in C_i,u\neq v} w_{(u,v)} < \infty \tag{4}$$

In the following, the argument of the minimization problem (3, left) is also referred to as the *cost* $C(P)$ of partitioning P. With the different options of using existing conceptual models, the designer has an expansive possibility of control. By choosing relevant model aspects, the whole method can be influenced according to individual preferences. On this basis, automated methods are to be used to solve the optimization problem and derive a set of business components that fulfills the target principles as good as possible. In contrast to manual methods, which strongly depend on the designer's competence, automated methods strive to detect the best possible solution. In the introduced approach, the designer has a second possibility to actively affect the derivation, though. Through the introduction of weights, (s)he can influence the optimization method, which is operating on the sets of elements and relationships. Primarily, weights could be assigned to certain types of relationships (e.g. *creating* an information object in a business process step could be weighted higher than *reading* it).

On the one hand, as the weights have a strong impact on the optimization results, the designer has a powerful possibility to influence the automated partitioning. On the other hand, however, a lot of experience is still necessary to derive consistent weights that reflect the project's goals. Therefore, decision support techniques are used to assist the designer when denoting preferences. Though, whenever such methods are used to evaluate different possibilities as basis for an optimized systems design, they carry the risk that slight changes in preferences will result in significantly different solutions (Zhu et al. 2005). Therefore, it has to be examined

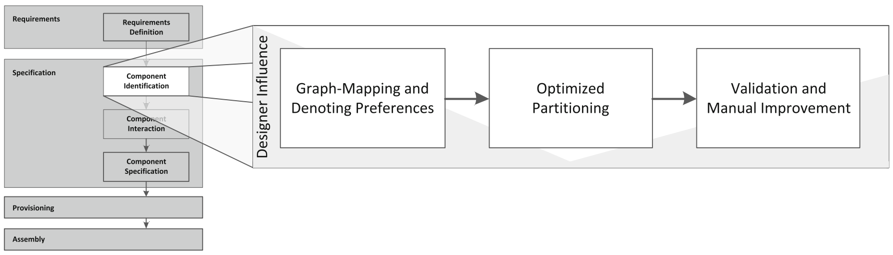


Fig. 3 Sub-steps of the *Component Identification* phase and the designer's influence

to what extent a change in preferences will still lead to equal results. For this, the introduced method allows simulating variations of weights and confronts the designer with the respective optimization results so that the influence of weights on the robustness of the derived component structure can be projected.

These considerations lead to the specification of a derivation method, which is organized in three phases (Fig. 3). The designer's influence varies from phase to phase. In the beginning of the *Graph-Mapping and Denoting Preferences* phase (s)he performs a reflective selection of elements of conceptual models which are to be considered and, thereby, has a high influence on the later outcome. During the determination of relationship weights, the designer is guided by decision support methods. Afterwards, the *Optimized Partitioning* is derived with automated optimization algorithms providing little to no possibility for interaction. This guarantees a high quality solution with respect to the minimizing coupling–maximizing cohesion principle even for large problems. The results of the second phase, however, ought to be reflected by the designer in controlled analyses. Together with the possibility for manual changes, this constitutes the final *Validation and Manual Improvement* phase.

4.2 Graph-mapping and denoting preferences

While many related approaches are based on structured matrix operations, newer approaches tend towards automated, graph-based methods (cf. Sect. 2.2). There are several reasons for this development, among which the most important are the availability of existing optimization methods and the possibility to include various information into the graph.

In general, a graph $G = (V, E)$ is defined through a finite, nonempty set of vertices V and a set of edges E. An edge $e = (u, v)$ is further defined as the connection between its adjacent nodes $u, v \in V$. In the case of a weighted graph, each edge e additionally has an assigned weight w_e (here: $w_e \geq 0$). In the context of this work, we furthermore define the three amendments: (i) any two nodes u and v might be connected with more than one direct edge $e_{(u,v)}$; (ii) vertices and edges can be of different static types and subtypes; (iii) vertices and edges might be marked as *external*. These additions are necessary to allow a mapping of the information contained in conceptual models onto a graph. Analogous graph definitions have repeatedly been used for the analysis of business processes in various contexts (e.g. Dijkman et al. 2009; Ouyang et al. 2009). These approaches commonly map elements from a business process diagram onto nodes, and control flows as well as other relations between elements onto edges.

For illustration purposes, we will discuss the procedure for the most common attributes of business processes and data models using Activity Diagrams and Class Diagrams of the Unified Modeling Language (UML) as examples (OMG 2005). Nevertheless, additional aspects could be used as well. Each business process model contains several actions or business process steps (BPS) which together describe a set of activities. Additionally, inputs and outputs of the process steps—here: solely in shape of information objects (IO), as material resources are not accounted for—are commonly contained in a process model. Finally, process steps are usually assigned to one or more actors. Actors could be computer systems, but also roles or departments in an organization are possible. The latter are relevant, as all business functions associated with one actor are likely to be thematically interrelated. Out of the set of elements, business process steps as well as information objects are mapped onto the nodes of a graph with a corresponding node type BPS or IO. This type attribute can be used in optimization methods to ensure that every business component contains at least one process step to control any included information objects. Furthermore, a node can be marked as *external*, if it is part of an existing or planned component which cannot or should not be altered by the designer (e.g. legacy systems or different projects). In contrast to business process steps (functionality) and information objects (resources), which have a direct equivalent in the implemented components (Szyperski et al. 2002), actors are not mapped onto nodes. Instead, all nodes of business process steps performed by a certain actor are connected through edges.

In a business process model, the order of single process steps is determined by the control flow. The resulting dependencies are mapped onto edges between the corresponding BPS nodes and, hence, are of the type BPS–BPS. Additionally, a process model describes the data flow between process steps so that each step can have input and output information objects. These relationships are mapped onto edges between BPS and IO nodes (edge type BPS–IO). Finally, entity relationships between information objects in a data model are mapped onto corresponding edges, as well (edge type IO–IO). The coarse-grained classification of edges with general types is further refined to resemble different intentions of relationships. Therefore, each edge can be assigned a certain subtype. Typical subtypes are e.g. function calls and the previously discussed actors for BPS–BPS relationships; create, read, update and delete for BPS–IO relationships (cf. the common CRUD operations); as well as related-to and state-of for IO–IO relationships. Weights are assigned to the edges based on their types and subtypes later on. As mentioned before, additional edge and node (sub-)types could be used as well, if necessary. Thus, our graph-based approach is highly flexible with respect to the designer's and project's demands.

Once model selection and graph mapping are done, the designer has to choose weights for each edge (sub-)type. However, due to the large amount of possible alternatives, it is oftentimes impractical for designers to report their complete preferences at once (Conitzer 2009). As the utilized weights have a strong impact on

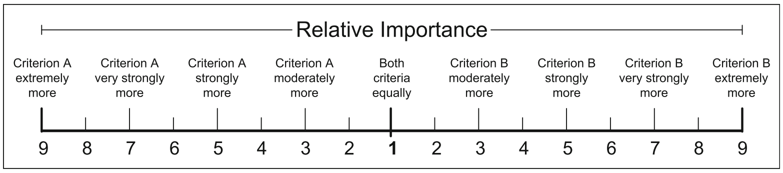

Relative Importance
Criterion A extremely more
Criterion A very strongly more
Criterion A strongly more
Criterion A moderately more
Both criteria equally
Criterion B moderately more
Criterion B strongly more
Criterion B very strongly more
Criterion B extremely more
9 8 7 6 5 4 3 2 1 2 3 4 5 6 7 8 9

Fig. 4 Standard AHP-Scale (cf. Saaty 1980)

the following optimization procedures, decision support methods are required to guide the designer towards a rational decision (Zhu et al. 2005). The Analytic Hierarchy Process (AHP) enables such a systematic derivation through a decomposition of the decision making problem into smaller subproblems which require less information and, hence, reduce the decision complexity (Saaty 1980). Using pairwise comparisons of different edge (sub-)types, the AHP forces the designer to concentrate on the comparative evaluation of two edge types at a time, instead of ranking all of them at once (Conitzer 2009). Additionally, the AHP reduces the cognitive load through phrasing the comparisons in natural sentences and verbalizing the answer scale (Guo and Sanner 2010). The ratio scale from 1 to 9 utilized for answers has also shown to be most effective for comparing different properties (Saaty 1980; Harker and Vargas 1987; Miller 1956). A typical question that has to be answered by the designer in an AHP is for instance: *Compared to (A) the creation of information objects in business process steps, how important is (B) the usage of information objects in business process steps? Mark the relative importance on the AHP-scale* (Fig. 4).

Such pairwise comparisons have to be evaluated by the designer for all types of edges. Furthermore, for each general edge type the AHP has to be repeated on a second level for all encompassed subtypes. The number of necessary pairwise comparisons for each group of edge types is defined as $\#Comparisons = \frac{t \cdot (t-1)}{2}$, where t resembles the number of different edge types in the group. If e.g. the designer further distinguishes between create, read, update, and delete relationships for the general edge type BPS–IO, the number of pairwise comparisons for these subtypes is $\frac{4 \cdot (4-1)}{2} = 6$. The number of comparisons is independent from the number of nodes and edges in the graph of a specific application scenario, but solely depends upon the generally relatively small number of different edge types. The comparison results are summarized into a matrix. Afterwards, impact factors for each type and subtype can be derived through eigenvalue calculations. For the exact definition we refer to Saaty (1980). Based on the impact factors of types and subtypes, the exact weights of all edges are derived as product of both.

One drawback of decomposing the decision making problem into pairwise comparisons is the possibility of inconsistencies between answers to different questions. E.g. if criterion A is moderately more important than criterion B, which itself is moderately more important than criterion C, then, to be consistent, A should be strongly more important than C. Therefore, it has to be verified that no contradictory choices have been made by the designer. For this, the *consistency* *ratio CR* of the matrix is calculated and used as a first indicator for problems. Only in some isolated cases, additional measures have to be examined (Kwiesielewicz and van Uden 2004). We adopt the common benchmark that if the *CR* is less than the critical value of 0.1 the results of the AHP are considered as fully usable with only minimal inconsistencies remaining (Saaty 1980). Otherwise, the designer would have to reevaluate the choices made.

4.3 Optimized partitioning

The derivation of an optimally designed IS is not an easy task, as an enormous amount of possible solutions exists. Likewise, identifying the partitioning of a graph which best suffices the defined requirements is known to belong to the class of *NP-complete* problems (Garey et al. 1974). Hence, any direct calculation of this partitioning by comparing all combinations is unreasonable for non-trivial problems. Nevertheless, heuristics can be used to identify the best possible solution with reasonable effort, which is, however, not necessarily the best overall solution. A common way to approach an optimization problem is the usage of combined opening and improving heuristics. While the earlier one identifies a possible starting solution, the latter uses this partitioning as the basis for further improvements. Since the starting solution is usually found to be a local optimum, the second heuristic has to provide some capability to pass local optima and enable solutions relatively close to the global optimum. Apparently, the improving heuristic can generally be applied to starting solutions of various sources (e.g. previous system designs, by-hand solutions, or results of other improving heuristics). In the following we will introduce a specifically developed greedy heuristic (Cormen et al. 1990) to generate the starting solution and an adoption of the partitioning algorithm from Kernighan and Lin (1970) for the subsequent improvement.

Greedy heuristics in general are rather straightforward procedures, which take the most promising action in each step without considering any further implications. Hence, as local optima are always taken, the resulting partitioning is usually not the global optimum. This practice can be compared to playing chess with only the upcoming step in mind. Our *Start Partition Greedy* heuristic aims to assemble closely related nodes into the same business component by examining relationships in the order of their individual strength. Its course of action favors the combination of tightly interconnected nodes and, thus, provides an implicit support to maximize the cohesion of created business components. The resulting initial component structure, therefore, can be used as a promising starting solution for further refinement. A necessary basis of the heuristic is an ordering of the edges $e \in E$ of the graph G as a priority queue (*PQ*).

Definition 6 (*Priority queue*) A priority queue sorts its elements according to their priority. The priority p_{e_i} of an edge e_i is higher than for e_j ($i \neq j$), if $w_{e_i} > w_{e_j}$. The opposite situation ($w_{e_i} < w_{e_j}$) is handled equivalently. In case of $w_{e_i} = w_{e_j}$, the respective weights of the adjacent nodes are to be compared. Here, the weight of a node $v \in V$ is defined as the sum of weights of all adjacent edges: $\sum_{e_{(u,v)} \in E | u \in V, u \neq v} w_{e_{(u,v)}}$.

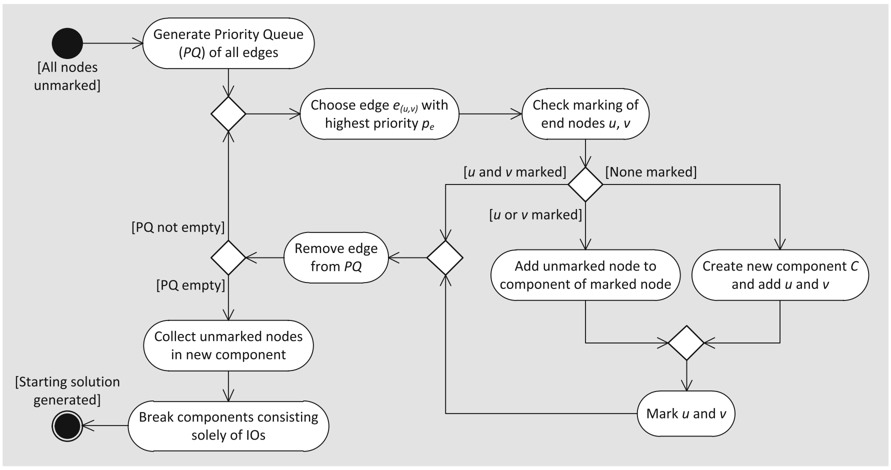

[All nodes unmarked]
Generate Priority Queue (*PQ*) of all edges
Choose edge $e_{(u,v)}$ with highest priority p_e
Check marking of end nodes *u*, *v*
[*u* and *v* marked]
[None marked]
[*u* or *v* marked]
[PQ not empty]
Remove edge from *PQ*
Add unmarked node to component of marked node
Create new component *C* and add *u* and *v*
[PQ empty]
Collect unmarked nodes in new component
[Starting solution generated]
Break components consisting solely of IOs
Mark *u* and *v*

Fig. 5 Start Partition Greedy heuristic

During the Start Partition Greedy heuristic, the *PQ* is processed iteratively. In each step, the edge on top of the PQ (i.e. the edge with the highest priority) is taken and its nodes are distributed to existing or new components (Fig. 5). At this point, an interesting opportunity of the method is the fact that the designer does not have to decide on the number of business components in the beginning, as it is solely determined by the heuristic. As a last step, business components consisting solely of IO are resolved. With respect to the minimizing coupling principle, each affected IO node is assigned to the component, towards which it has the closest relation to.

Kernighan and Lin (1970) were the first to define an efficient *improving heuristic* for an existing partitioning of a graph. To date, the original algorithm has been repeatedly adopted to different areas of application. The basic principle is an exchange of an equal number of nodes between two components to improve their respective partitioning costs. This two-components-optimization heuristic can be applied to the whole graph, by repeatedly examining pairs of components $(C_i, C_j)_{i \neq j}$, until no further improvements can be achieved. In so doing, trivial solutions to the optimization problem in favor for one conflicting partitioning goal (i.e. one single business component or a large amount of business components containing only one BPS node each) can be avoided. Again, to ensure conformance with the definition, components consisting solely of information objects are to be resolved in the heuristic.

The main step of the heuristic examines the exchange of nodes between two components *A* and *B*. For an efficient implementation of the algorithm, the following definitions are required.

Definition 7 (*Partitioning costs*) The costs $C(A, B)$ of a partitioning into two components *A* and *B* are defined equivalently to $C(P)$ from above:

$$C(A,B) = \sum_{a \in A, b \in B} w_{e_{(a,b)}} \tag{5}$$

Furthermore, each node $a \in A$ has defined external costs $E(a)$, internal costs $I(a)$, and a resulting cost difference $D(a)$.

$$E(a) = \sum_{b \in B} w_{e_{(a,b)}} \tag{6}$$

$$I(a) = \sum_{a' \in A, a' \neq a} w_{e_{(a,a')}} \tag{7}$$

$$D(a) = E(a) - I(a) \tag{8}$$

External and internal costs, as well as the cost difference are equivalently defined for each node $b \in B$. The gain $g(a, b)$, which is realized by an exchange of nodes a and b between the components A and B, is defined in the following Lemma.

Lemma 1 (Kernighan and Lin 1970) *Let $a \in A$ and $b \in B$. An exchange of nodes a and b results in a gain of precisely:*

$$\begin{aligned} g(a,b) &= C(A_{old}, B_{old}) - C(A_{new}, B_{new}) \\ &= D(a) + D(b) - 2w_{e_{(a,b)}} \end{aligned} \tag{9}$$

with $A_{new} = \{A_{old} \setminus \{a\}\} \cup \{b\}$ and $B_{new} = \{B_{old} \setminus \{b\}\} \cup \{a\}$.

The pairwise examination of components in the procedure depicted in Fig. 6 makes use of this lemma. In each step, it is decided which nodes should be

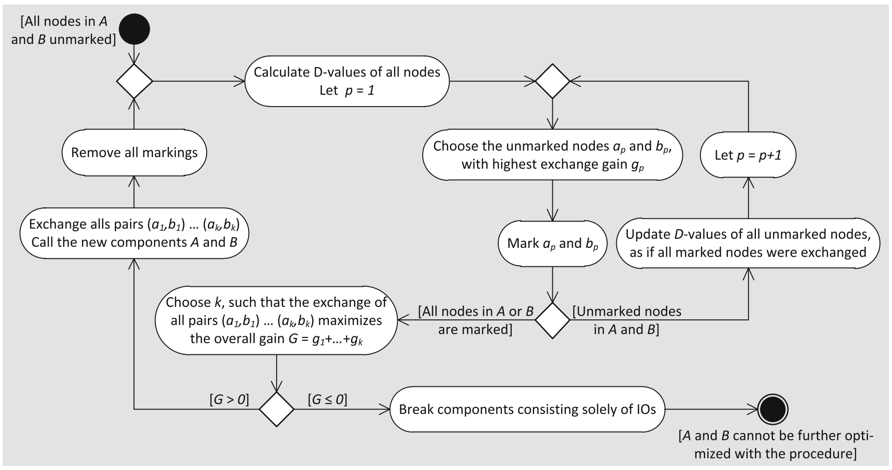

[All nodes in A and B unmarked]
Calculate D-values of all nodes Let p = 1
Choose the unmarked nodes a_p and b_p, with highest exchange gain g_p
Let p = p+1
Remove all markings
Mark a_p and b_p
Update D-values of all unmarked nodes, as if all marked nodes were exchanged
Exchange alls pairs (a_1,b_1) … (a_k,b_k) Call the new components A and B
Choose k, such that the exchange of all pairs (a_1,b_1) … (a_k,b_k) maximizes the overall gain $G = g_1+…+g_k$
[All nodes in A or B are marked]
[Unmarked nodes in A and B]
[G > 0]
[G ≤ 0]
Break components consisting solely of IOs
[A and B cannot be further optimized with the procedure]

Fig. 6 Two-components-optimization heuristic

exchanged for the highest gain. This gain can also be negative, which is a necessary requirement to resolve local optima (Kernighan and Lin 1970). The chosen pair of nodes is marked and treated as exchanged in the further steps. In the end, the final decision upon which exchanges are carried out depends on the enabled overall gain $G = \max_{k \leq n} \sum_{i=1}^{k} g_i$, where n is the number of nodes in the smaller component. If G is positive, the corresponding first k exchanges are fixed and the whole procedure is repeated. Once no further improvement is possible, the next pair of components will be examined until no more exchanges between any pairs of components are possible and, hence, the solution is optimal with respect to the algorithm.

An efficient recalculation of the $D(\hat{a})$ values for all unmarked nodes $\hat{a} \in A_{new}$ after an exchange of a and b is possible through: $D(\hat{a})_{new} = D(\hat{a})_{old} + 2w_{e_{(\hat{a},a)}} - 2w_{e_{(\hat{a},b)}}$. Respectively, the update for the nodes $\hat{b} \in B_{new}$ is done. Using this formula and an efficient sorting algorithm enables the two components optimization heuristic to achieve run-times of the order $\mathcal{O}(n^2 \log n)$, with n being the number of nodes in the smaller component.

For each solution, the algorithm additionally calculates the coupling and cohesion ratios of the component structure.

Definition 8 (*Coupling and cohesion ratios*) The coupling ratio $\overline{Cou(P)}$ of partitioning P is defined as the amount of coupling between components in relation to the overall amount of coupling between nodes in the graph. The cohesion ratio $\overline{Coh(P)}$ of partitioning P is defined respectively for cohesion within components.

$$\overline{Cou(P)} = \frac{Cou(P)}{\sum_{e \in E} w_e} \in [0, 1] \tag{10}$$

$$\overline{Coh(P)} = \frac{Coh(P)}{\sum_{e \in E} w_e} \in [0, 1] \tag{11}$$

The first ratio (10) can be used to measure the target achievement of the minimizing coupling principle. Lower values are generally favorable, as this indicates a lower amount of relationships between the business components. The second ratio (11) evaluates the maximizing cohesion target, accordingly.

4.4 Validation and manual improvement

As discussed earlier, the results of the automated derivation of business components should not be used without reflection. Especially the robustness of a derived optimization result is important for its long-term usability. The introduced optimization algorithm considerably depends on the initially chosen weights. Therefore, the influence of slightly different decisions during the AHP in the first phase and, thus, minimal changes in weights, needs to be examined (Zhu et al. 2005). A common way to test "the robustness of an optimal solution" and identify "critical values, thresholds or break-even values where the optimal strategy changes" are sensitivity analyses (Pannell 1997).

In such an analysis, the initial preferences are systematically varied and an optimized solution is calculated for every set of weights (Fiacco 1983). In several previous applications of our approach the variation of each impact factor from −50 to +50 % in steps of ten percentage points has shown to be sufficient. Note that the list of possible variations could be continued almost arbitrarily if every step size and all interactions are examined. The following comparison of different component structures against each other can generally be done in several ways. First of all, specific metrics can be used to evaluate certain aspects of the clustering (Chidamber and Kemerer 1994). In this scenario, especially the distribution of nodes into different business components ought to be examined, however. The fact whether groups of nodes stick together in various solutions or if they are randomly redistributed allows conclusions on the robustness of the derived component structure.

An examination of node distributions can either be performed by hand, or be supported through visualization tools. In our approach we utilize parallel coordinate plots to compare different optimization results.

Definition 9 [*Parallel coordinate plot* (Inselberg 1985)] On the plane with xy-Cartesian coordinates, and starting on the y-axis, N copies of the real line, labeled $x_1, x_2, \ldots, x_N$, are placed equidistantly (one unit apart) and perpendicularly to the x-axis. They are the axes of the parallel coordinate system and all have the same positive orientation as the y-axis. Each axis resembles one possible solution. For each record in a dataset the data-points are interconnected over all axes via horizontal lines.

This method is frequently used in statistical data analysis for "examining clustering in high dimensions" (Cook and Swayne 2007). Figure 7 shows how parallel coordinate plots are used during a sensitivity analysis in our approach. Here, each axis depicts a component structure derived from different weights. The marked axis in the middle of the plot resembles the original structure. The left side illustrates optimization results after a decrease and the right side after an increase of the controlled impact factor of up to ±50 %. Furthermore, each horizontal line resembles one node (i.e. a business process step or an information object from the

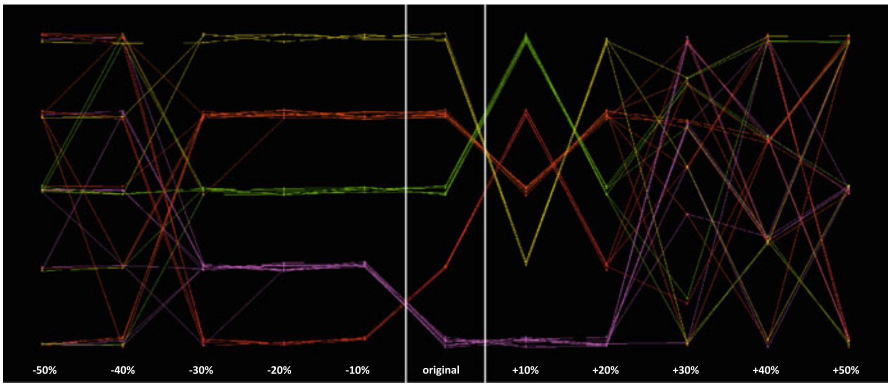


Fig. 7 Parallel coordinate plot in sensitivity analysis

input models) and shows to which component it is assigned to in the different optimization results. A separate plot is needed for every structured variation of one or more impact factors.

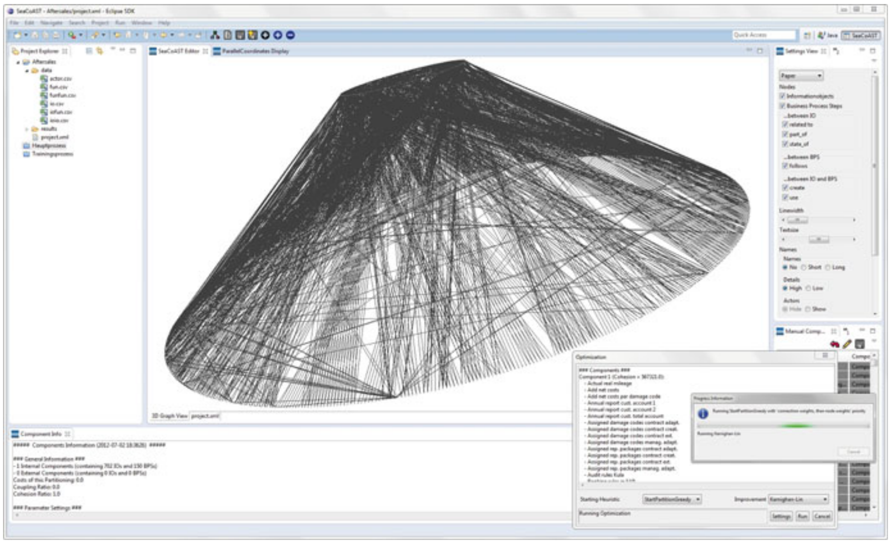
An examination of grouped lines reveals whether certain nodes keep together independently from the utilized weights. In the left part of Fig. 7 most groups of nodes stick together up to −30 %, which shows that in this example the results are rather robust towards the reduction of the impact factor(s). Note that different coordinates of groups of nodes in each variation are attributed to the internal numbering of components and have no significance regarding the component structure (cf. bottom lines at *original*-axis and −10 %-axis). On the right side of the example (Fig. 7) the results are stable until +20 % but show a sudden change at +30 %. Hence, −40 and +30 % mark the *breaking points* which lead to unstable results and significant changes in the derived component structures. As the robustness of a component-based structuring is an important prerequisite for its long-term applicability, any signs of unstable results ought to be carefully reflected during the design process. By enabling the designer to perform sensitivity analyses for relevant variations of each weight and to process conflicts that might emerge, the proposed derivation method systematically supports this task.

The designer also might use the partitioning as a profound starting point for further refinement through manual manipulations to consider information that was not mapped onto the graph (e.g. financial aspects). Finally, the designer has to examine the business functionality which each component performs and initiate the subsequent business component specification steps.

5 Tool support and application

For large problems, the introduced method cannot be applied by hand but has to be supported through appropriate software. To furthermore validate the general applicability of our approach, we have developed the *Services and Components Architecture Support Tool* (SeaCoAST). It implements the described method and, hence, supports the important design phase in the development process. Both—method and tool—have been successfully evaluated in several real-world use cases, ranging from small single application problems up to large IS development projects. The following describes its application to the after-sales processes of a German DAX-30 automobile manufacturer. All relevant data has been derived in a project with our industry partner (Eberhardt 2005). The complex business processes contain 150 process steps that are carried out by 21 different actors and affect 702 information objects.

The project's conceptual models (provided as Event-Driven Process Chains and Entity Relationship Diagrams) were mapped onto a graph as described in Sect. 4.2. The resulting initial structure of the graph—as it is visualized in SeaCoAST—is shown in Fig. 8 together with several information tabs and the optimization dialog. The top circle of nodes in the graph contains the business process steps and the bottom circle describes the information objects. Due to the size of the use case, details are only visible on the screen. The edges of the graph in this particular



```
##### Components Information (2012-07-02 18:36:28) #####

### General Information ###
- 1 Internal Components (containing 702 IOs and 150 BPSs)
- 0 External Components (containing 0 IOs and 0 BPSs)
Costs of this Partitioning: 0.0
Coupling Ratio: 0.0
Cohesion Ratio: 1.0

### Parameter Settings ###
```

```
### Components ###
Component 1 (Cohesion = 387321.0)
 - Actual real mileage
 - Add net costs
 - Add net costs per damage code
 - Annual report cust. account 1
 - Annual report cust. account 2
 - Annual report cust. total account
 - Assigned damage codes contract adapt.
 - Assigned damage codes contract creat.
 - Assigned damage codes contract ext.
 - Assigned damage codes manag. adapt.
 - Assigned rep. packages contract adapt.
 - Assigned rep. packages contract creat.
 - Assigned rep. packages contract ext.
 - Assigned rep. packages manag. adapt.
 - Audit rules Kulx
```

Starting Heuristic: StartPartitionGreedy
Improvement: Kernighan-Lin
Running Optimization

Fig. 8 SeaCoAST with a visualization of the initial graph and optimization dialogs

scenario are of the following subtypes. Within a process, one business process step *follows* another and each one is assigned to an *actor*. All process steps can *create* information objects, or *use* them during their execution. However, the input models did not differentiate between read and update relations. Furthermore, information objects can be *part of* larger objects, represent a certain *state of* another object, or simply be *related towards* each other.

After mapping the information from the conceptual models onto the graph, the optimization preferences had to be examined. Weights were derived by three experts based on an application of the introduced AHP in the *Denoting Preferences* step (Sect. 4.2). Among the experts involved in the discussion and voting, two were strongly familiar with the project and the underlying conceptual models. The third expert was acquired to provide an external view without knowledge of the specific project details. Even though the examined use case is rather large, there were only eight pairwise comparisons necessary to derive the impact factors (3 for the general edge types as well as 3, 1 and 1 for the respective subtypes). The resulting weights, as depicted in Fig. 9, reflect the following considerations. Among all edges, a special focus was placed onto those running between business process steps. In the project, functional relationships were considered the most important to achieve reasonable business components, as large network traffic would have presented a problem. In detail, *follow* relationships ($w = 0.432$) were regarded as more crucial than the assignment to *actors* ($w = 0.108$). Furthermore, information objects were to be placed preferably into the same component as the process step which *creates* ($w = 0.247$) them. The *usage* ($w = 0.050$) of information objects in process steps was considered to be less complex and hence low weights were assigned to those relationships. The structure of information objects was found to be less important than functional relationships. Entity relationships were further ordered according to

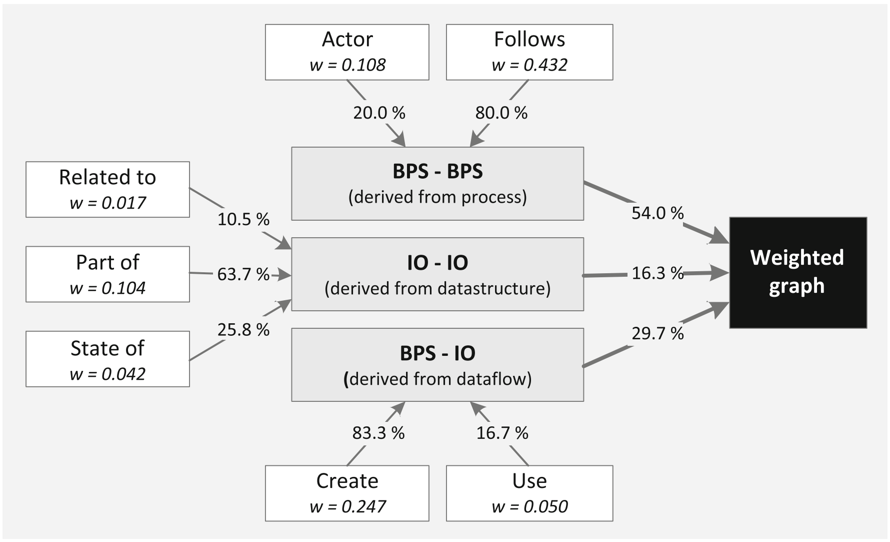


Fig. 9 Weights w and impact factors derived with AHP

the individual subtypes: *part-of* ($w = 0.104$) was viewed as a stronger relationship than *state-of* ($w = 0.042$) and also considered to be more important than *related-to* connections ($w = 0.017$). The consistency ratio CR of the AHP results varied between 0.000 and 0.033 with an overall value of 0.010. All of these are well below the critical value of 0.1 and, therefore, confirm that the derived weights contained negligible inconsistencies.

The weightings formed the basis for the automated *Optimized Partitioning* phase (Sect. 4.3). After running the Start Partition Greedy heuristic, the graph was partitioned into 26 components as depicted in Fig. 10. In the starting solution, component sizes varied between 6 and 93 nodes. The calculated partitioning costs were $C(P) = 30{,}606$ with a coupling ratio of $\overline{Cou(P)} \approx 0.533$ and a cohesion ratio of $\overline{Coh(P)} \approx 0.467$.

After applying the adopted improvement heuristic to the starting solution, the number of components was reduced to nine (Fig. 11). 140 iterations of the algorithm were needed to derive the improved partitioning. The respective costs were reduced to $C(P) = 5{,}722$ with $\overline{Cou(P)} \approx 0.100$ and $\overline{Coh(P)} \approx 0.900$. All identified components were carefully analyzed and interpreted by the designers from a business perspective and described according to their main tasks (Table 2).

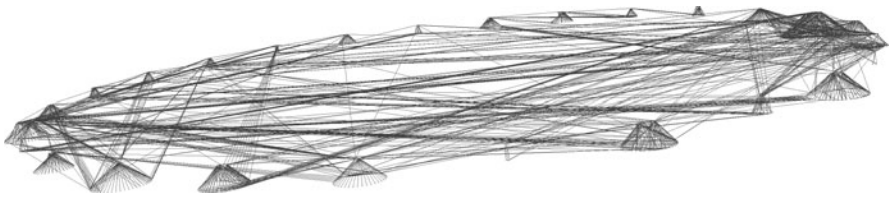


Fig. 10 Visualization of the result after Start Partition Greedy heuristic

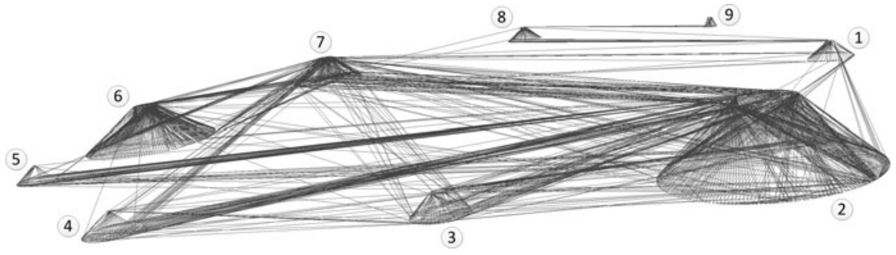


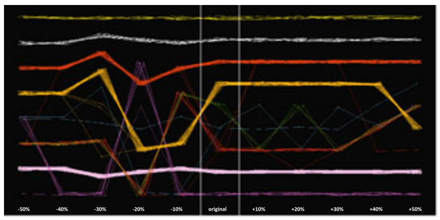
Fig. 11 Visualization of the result after start partition greedy and adopted Kernighan–Lin heuristics

Table 2 Summary of identified components

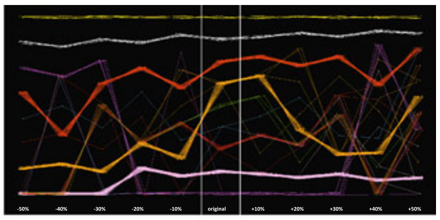
	Functional description	BPS-nodes	IO-nodes
1	Pricing	7	46
2	Customer management	58	185
3	Transaction processing	11	47
4	Reporting	2	51
5	Invoicing	3	42
6	Warranty and accommodating behavior check	28	167
7	Warranty and accommodating behavior processing	26	100
8	Portfolio definition	9	48
9	Market analysis	6	16

Manual corrections of the component structure were still possible. However, the designers did not make use of this possibility in the case, but rather performed an extensive sensitivity analysis with structured variations of the impact factors to validate the optimization results (Sect. 4.4). Figure 12 visualizes the effect of changes in the range from -50 up to $+50$ % to the original values for IO–IO, BPS–IO and BPS–BPS relationships (Fig. 9). Detailed examinations of the effect of changes to relationship subtypes have been performed as well, but are omitted here in the interest of brevity. As can be seen in the parallel coordinate plots, all components are fairly stable towards variations in the edge weights. This is especially true for variations to the impact of IO–IO relationships. Mainly, changes only affect a few rare BPS nodes that might be assigned to different components, due to low numbers of weak connections. However, it can also be seen that strong deviations of the impact factors of BPS–IO and BPS–BPS relationships influence the optimized solution when certain boundaries are passed. For BPS–IO relationships a change of $+20$ % or higher causes a sudden change in the component structure, which can be explained with the fact that those relationships are not completely dominated by the weight of BPS–BPS relationships anymore. Similarly, a variation of the weight of BPS–BPS relationships of more than -30 % leads to a changing structure as well, as BPS–BPS relationship types now have less relative importance than BPS–IO relationships.

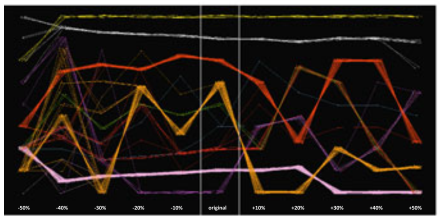
IO – IO



BPS – IO



BPS – BPS



Robustness

	Largely stable results	Breaking point	Unstable results
IO – IO	[-50%, +50%]	---	∅
BPS – IO	[-50%, +10%]	+20%	[+20%, +50%]
BPS – BPS	[-20%, +50%]	-30%	[-50%, -30%]

Fig. 12 Results and visualization of sensitivity analysis

Overall, the identified component structure is rather robust towards variations of up to ± 20 % and more from the initial preferences. The results of the AHP supported decisions are therefore invariant to slight changes and, hence, can be used without limitations. From the project's perspective, the sensitivity analysis confirmed that the derived components were tailored to the project's preferences and only a substantial change in goals would have resulted in a different IS design. In this context, the method itself and the results from the application of the introduced method were corroborated by our industry partners.

6 Theoretical evaluation and limitations

In addition to an examination of its general applicability, we theoretically compared the features of the method with those of the existing approaches discussed in Sect. 2.2. The classification of the introduced method according to the comparison categories is summarized in Table 3. The underlying business domain oriented understanding of business components (Definition 1) results in a top-down derivation procedure for components. With our method, each identified component comprises autonomous business functionality. Approaches based on different

Table 3 Theoretical classification of the introduced method

	Component definition	Degree of formalization	Direction	Identification strategy	Optimizing approach	Model views	Existing structures and system dependencies	Tool support	Quality assertions
Presented method	Domain-oriented	Algorithm	Top-down	Graph-based	■	Various	■	■	Evaluation

component understandings, on the other hand, are likely to neglect the business functionality of components (e.g. Cheesman and Daniels 2001). Furthermore, the method is readily usable in the early design phases of an IS development project, while especially technically-oriented methods are oftentimes inapplicable until detailed design decisions have been made (e.g. Jang et al. 2003; Rodrigues and Barbosa 2006). In Sect. 4, we have demonstrated the inclusion of several elements and relations from business process models and data models. In contrast to all other approaches (cf. Table 1), the method moreover can also be adapted to be used with various different input models. It can further be customized by the implementation of an individual weighting scheme, while most algorithmic procedures are not able to reflect any specific designer or project preferences (e.g. Jain et al. 2001; Meng et al. 2005; Cai et al. 2011). Yet, the designer is systematically guided throughout all steps of the method with detailed instructions, while several other approaches simply rely on general recommendations (e.g. Levi and Arsanjani 2002; Cui and Chae 2011). Furthermore, none of the examined approaches utilizes comparable decision support methods to provide additional assistance for the designer. Overall, our approach describes a reflective method in the sense of Definition 3.

However, there are also some limitations regarding the applicability of our approach. The proposed method is especially dependent on the level of detail that is provided with the conceptual models which serve as input. In particular, it is required that the business functions and information objects to be implemented are listed as individual elements of the input models. Due to this constraint, the design of the system becomes even more dependent on the quality of the conceptual models that have to be created initially (Barjis 2008). A second limitation that the method shares with all other existing approaches is the lack of a comprehensive empirical comparison regarding the results of different derivation techniques. To date, comparisons are restricted to theoretical evaluations due to different prerequisites (cf. Table 1) and the unavailability of developed tools and standard test-cases (Birkmeier and Overhage 2009). In future research we, therefore, plan to initiate an open project, which provides a large set of test models for business component derivation, with the goal to enable an empirical comparison of different approaches.

7 Conclusions

In this paper, we have presented a method to systematically derive business components from conceptual models. In particular, we described the partitioning strategy of minimizing coupling and maximizing cohesion together with its corresponding managerial goals as a customizable multi-criteria decision making problem that can be solved according to the preferences of the designer. Thereby, the presented method allows the designer to decide which elements of conceptual models and which relationships between them should be considered for the partitioning. Furthermore, the designer has to state design preferences using a rational procedure. Based on the model elements, relationships, and the preferences defined by the designer, a partitioning is generated and systematically optimized. The presented method finally supports an evaluation of the resulting partitioning for its stability against changing preferences.

This evaluation allows the designer to identify and process emerging conflicts that may lead to a different partitioning. To demonstrate how the method is applied in practice, we introduced the SeaCoAST tool and discussed its application in a complex use case taken from industry. Project partners attested the method to provide an "interesting opportunity to balance technically optimal IS design with individual project requirements" while "supporting the designer but not suppressing him".

The results of our research have implications for both practice and academia. For practice, they show how the partitioning of a business information system can be achieved using a rational procedure that can be customized according to project-specific requirements. In contrast to other methods, designers can hence adjust the partitioning procedure to resemble their assumptions about the criteria that should drive the partitioning. Possible clashes between the Weltanschauungs of the method creator and the method user, which lead to undesirable results when using development methods (Omland 2009), can so be minimized. The sensitivity analysis of the obtained results can, furthermore, help to clarify the existing design preferences. Depending on the robustness of the created partitioning, designers can use the evaluation results to rethink their preferences as the development process often "is driven by certain more or less clear objectives" (Mathiassen 1998). With the presented tool, it is furthermore possible to efficiently apply the method in larger projects, where a considerable amount of components will have to be derived. In such projects, the application of the presented method will likely lead to a partitioning that is better optimized for the autonomy of its constituent components than manual approaches. Especially the right degree of autonomy of business components is crucial to achieve the aspired goals of flexibility, reusability and reduced time to market when building upon a component-based approach, though.

For academia, our results show how the derivation of business components from conceptual models can be expressed in a way that ensures active participation and influence of the designer while the partitioning is still optimized systematically. The presented method provides a compromise in the current spectrum of fully automated methods, which do not account for any influence or control, and manual approaches, which do not guarantee for a sufficiently optimized partitioning. Furthermore, we suggest several types of conceptual model elements and relationships which can be taken into account during the derivation of components. With the presented method, we address the persisting research gap of how to partition a design space into business components, which is one of the few challenges that yet remain to be solved in the area of component-based IS development. In contrast to existing approaches, the presented method provides support for a reflected, customized partitioning. It hence serves as a contribution to what Mathiassen (1998) called "reflective systems development". During the application of the method, especially this aspect turned out to positively impact developer acceptance.

Besides further evaluating the presented method in field studies with industry partners, future research will also focus on determining best practices for weightings that were applied successfully in projects with comparable preferences. These best practices can serve as benchmark for designers to analyze their own, situational weightings against. Finally, we will tailor our method to the closely related service-oriented computing discipline. In that context, we currently work on systematically deriving services from conceptual models in order to contribute to the forming of specialized development methodologies that support the service-oriented design of IS.

References

Albani A, Keiblinger A, Turowsi K, Winnewisser C (2003) Domain based identification and modelling of business component applications. In: Kalinichenko LA, Manthey R, Thalheim B, Wloka U (eds) Proceedings of the 7th East European conference on advances in databases and information systems. Springer LNCS 2798, Berlin, pp 30–45

Albani A, Overhage S, Birkmeier D (2008) Towards a systematic method for identifying business components. In: Chaudron MRV, Szyperski C, Reussner R (eds) Component-based software engineering, 11th international symposium. Springer LNCS 5282, Berlin, pp 262–277

Barjis J (2008) The importance of business process modeling in software systems design. Sci Comput Program 71:73–87

Bass L, Clements P, Kazman R (1998) Software architecture in practice. Addison-Wesley, Upper Saddle River

Baster G, Konana P, Scott JE (2001) Business components—a case study of Bankers Trust Australia Limited. Commun ACM 44(3):92–98

Birkmeier DQ, Overhage S (2009) On component identification approaches—classification, state of the art, and comparison. In: Lewis GA, Poernomo I, Hofmeister C (eds) Component-based software engineering, 12th international symposium. Springer LNCS 5582, Berlin, pp 1–18

Birkmeier DQ, Overhage S (2012) A semi-automated approach to support the architect during the generation of component-based enterprise architectures. In: Proceedings of the 20th European conference on information systems, Barcelona, pp 1–12

Birkmeier DQ, Kloeckner S, Overhage S (2009) A survey of service identification approaches: classification framework, state of the art, and comparison. Enterp Model Inf Syst Archit 4(2):20–36

Blois APTB, Werner CML, Becker K (2005) Towards a components grouping technique within a domain engineering process. In: Proceedings of the 31st EUROMICRO conference on software engineering and advanced applications, IEEE Computer Society, pp 18–27

Brown AW (2000) Large-scale, component-based development. Object and Component Technology Series. Prentice Hall, Upper Saddle River

Cai ZG, Yang YH, Wang XY, Kavs AJ (2011) A fuzzy formal concept analysis based approach for business component identification. J Zhejiang Univ Sci C 12(9):707–720

Cheesman J, Daniels J (2001) UML components: a simple process for specifying component-based software. The Component Software Series. Addison-Wesley, Upper Saddle River

Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493

Chidamber SR, Darcy DP, Kemerer CF (1998) Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Trans Softw Eng 24(8):629–639

Conitzer V (2009) Eliciting single-peaked preferences using comparison queries. J Artif Intell Res 35:161–191

Cook D, Swayne DF (2007) Interactive and dynamic graphics for data analysis—with R and GGobi. Use R! Springer, New York

Cormen TH, Leiserson CE, Rivest RL (1990) Introduction to algorithms. The MIT electrical engineering and computer science series. MIT Press, Cambridge

Cui JF, Chae HS (2011) Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. Inf Softw Technol 53(6):601–614

Dijkman R, Dumas M, Garca-Bauelos L (2009) Graph matching algorithms for business process model similarity search. In: Dayal U, Eder J, Koehler J, Reijers H (eds) Business process management. Springer LNCS 5701, Berlin, pp 48–63

D'Souza DF, Wills AC (1999) Objects, components, and frameworks with UML: the catalysis approach. The Object Technology Series. Addison-Wesley, Upper Saddle River

Eberhardt A (2005) Konzeption einer IT-Lösung zur Unterstützung des kombinierten Servicevertrags- und Garantieprozesses am Beispiel der DCAG (in German). Diploma thesis, University of Augsburg

Fiacco AV (1983) Introduction to sensitivity and stability analysis in nonlinear programming. Mathematics in science and engineering. Academic Press, New York

Fitzgerald B, Russo NL, Stolterman E (2002) Information systems development: methods in action. McGraw-Hill, London

Ganesan R, Sengupta S (2001) O2BC: A technique for the design of component-based applications. In: Proceedings of the 39th international conference and exhibition on technology of object-oriented languages and systems, pp 46–55

Garey MR, Johnson DS, Stockmeyer L (1974) Some simplified NP-complete problems. In: Proceedings of the 6th annual ACM symposium on theory of computing. ACM, pp 47–63

Gorton I (2011) Essential software architecture, 2nd edn. Springer, Heidelberg

Gui G, Scott PD (2008) New coupling and cohesion metrics for evaluation of software component reusability. In: Proceedings of the 9th international conference for young computer scientists. IEEE Computer Society, pp 1181–1186

Guo S, Sanner S (2010) Real-time multiattribute Bayesian preference elicitation with pairwise comparison queries. In: Proceedings of the 13th international conference on artificial intelligence and statistics, JMLR: W&CP, Chia Laguna Resort, Sardinia, Italy, vol 9

Hadar I, Soffer P (2006) Variations in conceptual modeling: classification and ontological analysis. J Assoc Inf Syst 7(8):569–593

Harker PT, Vargas LG (1987) The theory of ratio scale estimation: Saaty's analytic hierarchy process. Manag Sci 33(11):1383–1403

Herzum P, Sims O (2000) Business component factory: a comprehensive overview of component-based development for the enterprise. OMG Press Books in Print. Wiley, New York

Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MISQ 28(1):75–105

Hopkins J (2000) Component primer. Commun ACM 43(10):27–30

IBM Corporation (1984) Business systems planning: information systems planning guide. Technical report ge20-0527-4, International Business Machines Corporation

Iivari J (1995) Object-orientation as structural, functional and behavioral modelling: a comparison of six methods for object-oriented analysis. Inf Softw Technol 37(3):55–163

Iivari J (2007) A paradigmatic analysis of information systems as a design science. Scand J Inf Syst 19(2):39–64

Iivari J, Hirschheim R, Klein HK (1998) A paradigmatic analysis contrasting information systems development approaches and methodologies. Inf Syst Res 9(2):164–193

Inselberg A (1985) The plane with parallel coordinates. Vis Comput 1(2):69–91

Jain H, Chalimeda N, Ivaturi N, Reddy B (2001) Business component identification—a formal approach. In: Proceedings of the 5th IEEE international conference on enterprise distributed object computing, IEEE Computer Society, pp 183–187

Jang YJ, Kim EY, Lee KW (2003) Object-oriented component identification method using the affinity analysis technique. In: Proceedings of the 9th international conference on object-oriented information systems

Kang KC, Kim S, Lee J, Kim K, Shin E, Huh M (1998) FORM: a feature-oriented reuse method with domain-specific reference architectures. Ann Softw Eng 5:143–168

Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Technol J 49:291–307

Kim SD, Chang SH (2004) A systematic method to identify software components. In: Proceedings of the 11th Asia-Pacific software engineering conference, pp 538–545

Kohlborn T, Korthaus A, Chan T, Rosemann M (2009) Identification and analysis of business and software services—a consolidated approach. IEEE Trans Serv Comput 2(1):50–64

Kwiesielewicz M, van Uden E (2004) Inconsistent and contradictory judgements in pairwise comparison method in the AHP. Comput Oper Res 31:713–719

Lee EA, Xiong Y (2001) System-level types for component-based design. In: Henzinger TA, Kirsch CM (eds) Embedded software, 1st international workshop. Springer LNCS 2211, Berlin, pp 237–253

Lee SD, Yang YJ, Cho ES, Kim SD, Rhew SY (1999) COMO: A UML-based component development methodology. In: Proceedings of the 6th Asia-Pacific software engineering conference, pp 54–61

Levi K, Arsanjani A (2002) A goal-driven approach to enterprise component identification and specification. Commun ACM 45(10):45–52

Li W, Henry SM (1993) Object-oriented metrics that predict maintainability. J Syst Softw 23(2):111–122

Lim WC (1998) Managing software reuse—a comprehensive guide to strategically reengineering the organization for reusable components. Prentice Hall, Upper Saddle river

March ST, Smith GF (1995) Design and natural science research on information technology. Decis Support Syst 15(4):251–266

Mathiassen L (1998) Reflective systems development. Scand J Inf Syst 10(1&2):67–118

McDonald M, Begin J, Fortino S (2009) Meeting the challenge: the 2009 CIO agenda. Tech. rep., Gartner, Inc.

Meng FC, Zhan DC, Xu XF (2005) Business component identification of enterprise information system: a hierarchical clustering method. In: Proceedings of the IEEE international conference on e-Business engineering, pp 473–480

Meyer B (1997) Object-oriented software construction, 2nd edn. Prentice Hall, Upper Saddle River

Miller GA (1956) The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol Rev 63(2):81–97

OMG (2005) Unified modeling language specification: version 2. Revised Final Adopted Specification ptc/05-07-04, Object Management Group

Omland HO (2009) The relationships between competence, methods, and practice in information systems development. Scand J Inf Syst 21(2):3–26

Ouyang C, Dumas M, van der Aalst WMP, ter Hofstede AHM, Mendling J (2009) From business process models to process-oriented software systems. ACM Trans Softw Eng Methodol 19:2:1–2:37

Pannell DJ (1997) Sensitivity analysis of normative economic models: theoretical framework and practical strategies. Agric Econ 16:139–152

Parnas DL (1972) On the criteria to be used in decomposing systems into modules. Commun ACM 15(12):1053–1058

Rodrigues NF, Barbosa LS (2006) Component identification through program slicing. Electron Notes Theor Comput Sci 160:291–304

Saaty TL (1980) The analytic hierarchy process: planning, priority setting, resource allocation. McGraw-Hill, New York

Sharp J, Ryan S (2005) A review of component-based software development. In: Proceedings of the 26th international conference on information systems

Shaw M, Garlan D (1996) Software architecture: perspectives on an emerging discipline. Prentice Hall, Upper Saddle River

Smolander K, Rossi M (2008) Conflicts, compromises, and political decisions: methodological challenges of enterprise-wide e-business architecture creation. J Database Manag 19(1):19–40

Sugumaran V, Tanniru M, Storey VC (1999) Identifying software components from process requirements using domain model and object libraries. In: Proceedings of the 20th international conference on information systems

Szyperski C, Gruntz D, Murer S (2002) Component software—beyond object-oriented programming, vol 2. ACM Press; Addison-Wesley, New York

Takeda H, Veerkamp P, Tomiyama T, Yoshikawa H (1990) Modeling design processes. AI Mag 11(4):37–48

Topi H, Ramesh V (2002) Human factors research on data modeling: a review of prior research, an extended framework and future research directions. J Database Manag 13(2):3–15

Vaishnavi V, Kuechler B (2004) Design research in information systems. Online article, http://ais.affiniscape.com/displaycommon.cfm?an=1&subarticlenbr=279

Vitharana P, Zahedi F, Jain H (2003) Knowledge-based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis. IEEE Trans Softw Eng 29(7):649–664

Vitharana P, Jain H, Zahedi FM (2004) Strategy-based design of reusable business components. IEEE Trans Syst Man Cybern 34(4):460–474

Wallnau KC, Hissam SA, Seacord RC (2002) Building systems from commercial components. SEI series in software engineering, Addison-Wesley, Upper Saddle River

Wirth N (1971) Program development by stepwise refinement. Commun ACM 14(4):221–227

Yourdon E, Constantine LL (1979) Structured design: fundamentals of a discipline of computer program and system design. Prentice-Hall, Upper Saddle River

Zhu L, Aurum A, Gorton I, Jeffery R (2005) Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process. Softw Qual J 13:357–375