A SEMI-AUTOMATED APPROACH TO SUPPORT THE ARCHITECT DURING THE GENERATION OF COMPONENT-BASED ENTERPRISE ARCHITECTURES

Dominik Birkmeier
University of Augsburg

Sven Overhage
University of Augsburg

A SEMI-AUTOMATED APPROACH TO SUPPORT THE ARCHITECT DURING THE GENERATION OF COMPONENT-BASED ENTERPRISE ARCHITECTURES

Birkmeier, Dominik Q., University of Augsburg, Universitätsstraße 16, 86159 Augsburg, Germany, dominik.birkmeier@wiwi.uni-augsburg.de

Overhage, Sven, University of Augsburg, Universitätsstraße 16, 86159 Augsburg, Germany, sven.overhage@wiwi.uni-augsburg.de

Abstract

In today's volatile business environments, enterprises need to be able to flexibly adapt their information systems and add new functionality quickly. Component-based enterprise architectures promise to help solving these challenges by structuring information systems landscapes into modular business components. However, the derivation of business components from conceptual models still poses research challenges as current methods do not adequately involve the architect and his/her situational preferences. In this paper, we propose an advanced method that facilitates a systematic, reflected derivation of business components. The novel contribution of this paper thereby is to show (i) how the architect can define the derivation of business components from conceptual models as a multi-criteria decision-making problem according to his/her situational preferences and (ii) how the architect can systematically verify the stability of the derivation results. We demonstrate the feasibility of the proposed method by demonstrating its implementation as part of the SeaCoAST tool and applying it to the after-sales processes of a world-wide leading automobile manufacturer.

Keywords: Enterprise architecture, component-based structuring, IS landscape design, business component identification, design science.

1 Introduction

Today, enterprise architecture (EA) strategies have to fulfill a variety of challenges. Volatile economic conditions require enterprises to quickly adapt their information systems (IS) and deliver new functionality fast (McDonald et al., 2009). Additionally, they need to improve the cost-efficiency of IS landscapes by consolidating the implemented IS operations (McDonald et al., 2009). In this context, component-based structuring approaches promise to bring many advantages (Herzum and Sims, 2000, Vitharana, 2003). They partition IS landscapes into loosely coupled parts of business functionality such that IS can be more easily adapted to changes and augmented with new functionality (Sharp and Ryan, 2005). As the elementary landscape elements – the business components – can be reused in different application scenarios, such a structuring also helps consolidating the implemented IS operations (Sharp and Ryan, 2005). As a result, component-based structuring approaches attracted great attention in research and industry.

With the increasing significance, methodologies to support the component-based development paradigm have been introduced (Herzum and Sims, 2000, Brown, 2000). They provide methods for the design, implementation, and assembly of business components (Szyperski et al., 2002). Nevertheless, the component-based paradigm continues to pose research challenges. Especially the question of how to systematically structure the functional scope of IS landscapes into business components yet has to be answered (Birkmeier and Overhage, 2009). On the one hand, fine-grained business components which each implement a specific functionality are easier to reuse than coarse-grained business components with a complex functional scope (Szyperski et al., 2002). The structuring of IS landscapes into fine-grained business components thus helps consolidating the provided IS operations and avoiding redundancies. On the other hand, a fine-grained structuring complicates the assembly of new IS and the adaption of existing IS to changing requirements since the number of interfaces between components increases. The IS landscape hence becomes more complex to maintain (Szyperski et al., 2002). Architects will have to strive for an optimal functional scope of business components that balances reusability with the number of interfaces to other components.

Although methods have been proposed to support the architect during the structuring of IS landscapes, determining an optimal functional scope of business components remains a cumbersome process. One reason is that these methods are usually based on implicit assumptions about the design preferences that determine an optimal functional scope of business components. Differences in the Weltanschauungs of the method's creator and user hence result in a way of using the method that differs from the creator's intentions (Omland, 2009). Moreover, many of the proposed methods build upon fully automated procedures to determine an optimized functional scope of business components. However, architects usually benefit more from methods "that help to identify and process the emerging conflicts" than from methods that merely calculate some technically perfect solution (Smolander and Rossi, 2008). In this paper, we present an advanced method that supports a systematic, reflected derivation of business components. It is based upon a rational procedure that assists architects in determining an optimized functional scope of business components according to their individual design preferences. The method takes conceptual models as input to identify distinct business functionalities and assign them to business components. The novel contribution of our approach thereby is to show *(i) how the architect can define the derivation of business components from conceptual models as a multi-criteria decision-making problem according to his/her situational preferences and (ii) how the architect can systematically verify the stability of the derivation results.*

Next, we discuss related work to highlight the research gap, followed by the research method (section 3). We then present our approach to systematically support the architect during the derivation of business components (section 4). In section 5, we describe an implementation of our method as part of the SeaCoAST tool and show its application in an industry case. Finally, we describe limitations and implications.

2 State of the Art and Related Work

In parallel with the formation of component-based development methodologies, approaches to support the derivation of software components in general and business components in particular have been proposed in literature. To generate an optimized structuring of the design space, these approaches usually follow the established design principle of minimizing coupling and maximizing cohesion (Parnas, 1972, Szyperski et al., 2002). Coupling for a component can be defined as "the extent to which elements within a component relate to the other elements, which are not in that component" (Vitharana et al., 2004). Cohesion of a component, furthermore, is "the extent to which its elements are interrelated" (Vitharana et al., 2004). At a first glance, the principle reflects technical considerations in the design. Yet, Chidamber et al. (1998) and Vitharana et al. (2004) have compiled how the technical features *coupling* and *cohesion* map to the general managerial goals in systems structuring: *cost effectiveness*, *customizability*, *reusability* and *maintainability*. Therefore, a "good" IS structuring (i.e. with cost effective, customizable, reusable components and low maintenance costs) aims to balance the two (conflicting) structuring goals: to minimize coupling between components, elements should be grouped into a component whenever they are processed together at some point. To maximize cohesion, a component should only contain elements which are all processed together.

Existing approaches especially differ in their support for a systematic procedure (Birkmeier and Overhage, 2009). They range from general recommendations that are considered during the derivation (Levi and Arsanjani, 2002, Sugumaran et al., 1999, Cui and Chae, 2011) to structured methods (Meng et al., 2005, Albani et al., 2008). In line with the literature, we define a method as "a coherent and systematic approach, based on a particular philosophy of systems development, which will guide developers on what steps to take, how these steps should be performed and why these steps are important in the development" (Fitzgerald et al., 2002). Extending a survey from Birkmeier and Overhage (2009), we therefore examine the approaches with respect to their underlying Weltanschauung and their support for a reflective process.

The approaches, first of all, differ in their understanding of components, which is a key influence factor on the results of an approach however (Birkmeier and Overhage, 2009). So-called business components, which are in the focus of this paper, implement a specific business functionality and can be broadly understood as a "software implementation of an autonomous business concept or business process" (Herzum and Sims, 2000). While such an understanding is focused on the conceptual functions that are provided, others emphasize technical aspects. Approaches that build upon a technical view of components oftentimes propose to analyze code and database schemas instead of conceptual, domain-oriented models to derive software components as higher-order design elements (Jang et al., 2003, Rodrigues and Barbosa, 2006). Many of the proposed business-oriented approaches utilize matrix analyses to derive components from domain or IT models (Lee et al., 1999, Ganesan and Sengupta, 2001, Kim and Chang, 2004). Most of them build upon the Business Systems Planning approach (IBM Corporation, 1984), which denotes individual business functions and processed business data as two dimensions of a matrix. In the matrix, relationships between business functions and data (such as create and use relationships) are charted. They serve as the basis to identify clusters according to the above design principle. Methods that build upon matrix analyses are limited regarding the information used to derive business components, though. Focusing solely on relationships between functions and data can compromise the IS landscape design 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, clustering analyses and graph-based methods have been proposed (Jain et al., 2001, Lee et al., 2001, Cai et al., 2011, Albani et al., 2008). While such techniques are basically able to work with arbitrary kinds of relationships, the proposed methods are usually restricted to pre-defined sets of relationships (Birkmeier and Overhage, 2009). Architects, hence, are not able to customize the methods according to the requirements of their projects and to the available information.

Furthermore, proposed approaches either specify the derivation of components as guidelines with several work steps which have to be performed manually (Ganesan and Sengupta, 2001, IBM Corporation, 1984, Lee et al., 1999) or introduce completely automated derivation procedures which cannot be influenced by the architect at all (Kim and Chang, 2004, Blois et al., 2005). Especially in industry-sized EA projects, manually deriving components based on matrix or clustering analyses burdens the architects with significant cognitive load, though. On the other hand, completely automated methods do not allow for a sufficient level of influence during the derivation process. Because the structuring of IS landscapes is a significantly creative process, "it is important that the individual developers engage their competencies" during component derivation, however (Omland, 2009, Fitzgerald et al., 2002). Therefore, semi-automated approaches integrate the architect in the derivation procedure (Albani et al., 2008, Cai et al., 2011), but without appropriate support and guidance (Table 1). Thus, it still strongly "depends on the knowledge of the designer if […] useful results can be achieved" (Birkmeier and Overhage, 2009).

Summing up, manual approaches strongly rely on the architect during the derivation of components. Among the analyzed approaches, only the automated ones are able to generate adequate optimized solutions with respect to the minimize coupling – maximize cohesion principle even for large projects, though.

Fully automated approaches reduce the possibility for human failures and the dependency on the individual architect's competence. However, such methods also hamper the chance to achieve outstanding solutions in IS structuring, as the architect's expertise and creativity are suppressed. Semi-automated approaches therefore integrate possibilities for customization into the preparation and the analysis phase. However, these approaches fail to support the architect with adequate guidelines (Table 1).

		IBM Corporation (1984)	Lee et al. (1999)	Sugumaran et al. (1999)	Ganesan and Sengupta (2001)	Jain et al. (2001)	Lee et al. (2001)	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)
	Optimizing approach	Yes	Yes	No	No	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
	Architect support	Partially guided	Unguided	Unguided	Unguided	Unguided	Partially guided	Partially guided	Partially guided	Partially guided	Unguided	Unguided	Unguided	Unguided	Partially guided	Unguided
Architect influence	Preparation	Low	None	High	High	None	High	High	---	Low	Low	---	None	High	High	Low
	Derivation	High	Low	High	High	None	None	High	Low	Low	High	Low	None	None	None	None
	Analysis	---	---	High	---	High	---	---	---	High	---	None	---	---	---	None

Table 1 Architect integration in related approaches

To address these shortcomings and contribute to the closure of this research gap, we propose a method that guides and supports the architect throughout the important derivation steps. Building upon the semi-automated method described in Albani et al. (2008) the *derivation* phase itself is automated to a considerable part to ensure a good structuring with respect to technical features and managerial goals. There will, nevertheless, be several guided possibilities during the *preparation* to influence the method to fit situational project preferences. We will furthermore describe a structured *analysis* to evaluate and reflect the 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). On the one hand, it is the rigor of constructing IT artifacts that distinguishes design science research from the practice of building IT artifacts (Iivari, 2007). The construction of the artifact hence has to follow a rigorous, scientific process. For the development of the presented method, we followed the *design cycle* introduced by Takeda et al. (1990), which defines the problem statement, solution concept, solution instantiation, and evaluation as milestones. To make the construction process that we followed transparent, we provide details about each stage of the 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. On the other hand, the constructed IT artifact has to represent an innovative solution to the existing knowledge base (Hevner et al., 2004). Hence, it has to be scientifically evaluated in order to prove that the research goals driving the artifact construction have been fulfilled (Iivari, 2007). Thus, we evaluated the presented method in two ways: to demonstrate the feasibility, we *implemented* the method in a tool and *applied* it to solve practical design tasks. We will provide details on the implementation and an application example later on.

4 Business Component Derivation

A large variety of information can be used as starting point for business component derivation methods, whereat this work is focused on conceptual models. Such a model "describes the users' world separately from any notation of the target software" (D'Souza and Wills, 1999). Thus, input can be provided from business process models (e.g. Business Process Modeling Notation diagrams), data models (e.g. Entity Relationship Diagrams) and function models (e.g. Function Decomposition Diagrams). For business component derivation, a *set of elements* from conceptual models is considered together with the *set of relationships* between the 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 e.g. control flows between process steps or entity relationships between information objects. However, it depends on the project and is furthermore up to the architect to decide which concrete models, elements and relationships are available and considered in the derivation process. Additionally, relationships between elements could be of different types and importance. To reflect the latter, an architect 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 $P = \{C_1, C_2, \dots, C_k\}$ of pairwise disjoint subsets C_i, such that the union of all components $\bigcup_{i=1}^{k} C_i$ matches the whole set. In this context, coupling and cohesion can be formalized into concrete measures.

Definition (Vitharana et al., 2004): *Measures of* coupling *(1) and* cohesion *(2) are defined as:*

$$(\mathbf{1}) \quad \mathrm{Cou(P)} = \tfrac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\notin C_i} w_{(u,v)} \ ; \qquad (\mathbf{2}) \quad \mathrm{Coh(P)} = \tfrac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\in C_i, u\neq v} w_{(u,v)}$$

Therefore, the minimizing coupling – maximizing cohesion principle depends on 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 on the overall structuring. Following the objective of minimizing coupling in a component structure, the target function of the optimization process, hence, can be stated as: *Minimize the sum of weights of all relationships between elements in different business components (3).* 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 (4).* This second goal resembles the maximizing cohesion principle.

$$(\mathbf{3}) \qquad \min_P \tfrac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\notin C_i} w_{(u,v)} \quad \Leftrightarrow \quad \max_P \tfrac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\in C_i, u\neq v} w_{(u,v)} \qquad (\mathbf{4})$$

Proof: *The equivalency follows from the fact that the sum of weights of all existing relationships (5) remains constant:*

$$(\mathbf{5}) \qquad \sum_{\forall u,v} w_{(u,v)} = \tfrac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\notin C_i} w_{(u,v)} + \tfrac{1}{2}\sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\in C_i, u\neq v} w_{(u,v)} < \infty$$

The argument of the minimization problem (3) is also referred to as the *cost* $C(P)$ of partitioning P. With the different options of using existing conceptual models, the architect has an expansive possibility of control. By choosing relevant aspects, the whole method can be influenced according to individual preferences. On this basis, automated methods can be used to 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 architect's competence, automated methods strive to detect the best possible solution. In the introduced approach, the architect still has a second possibility to actively affect the derivation, though. Through the introduction of weights, (s)he can influence an 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 architect has a powerful possibility to influence the automated structuring. On the other hand, a lot of experience is still necessary to derive consistent weights that reflect the project goals. Therefore, decision support techniques are utilized to assist the architect in denoting preferences. Though, whenever such methods are used to evaluate different possibilities as basis for an optimized structuring of IS landscapes, 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 to what extent a change in preferences will produce stable results. The introduced method envisions a simulated variation of weights and a confrontation of the respective optimization results, such that the influence of weights on the robustness of the component structure can be projected.

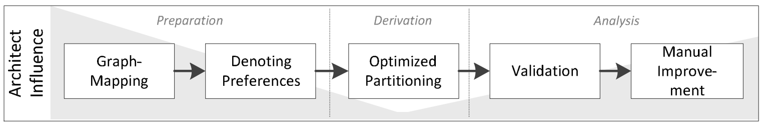


Figure 1. Architect influence in derivation method

These considerations lead to a derivation method, which is organized in three parts (Figure 1). The architect's influence varies from phase to phase. In the beginning (s)he chooses the elements of conceptual models which are to be *mapped onto a graph* and, thus, has a high influence on the later outcome. During the determination of relationship weights in the *Denoting Preferences* phase, the architect is guided through decision support methods. Afterwards, a partitioning is derived with *optimization* algorithms providing no possibility for interaction. This guarantees a high quality solution even for large projects with respect to the minimizing coupling – maximizing cohesion principle. The results, however, ought to be *validated* by the architect in controlled analyses, which might lead to some *manual changes*.

4.1 Graph-Mapping and Denoting Preferences

Business processes are frequently mapped onto graphs for various analyses. We have adopted the graph definition of Ouyang et al. (2009). For illustration purposes, we will discuss the procedure for the most common attributes of business processes and data models using UML Activity Diagrams and Class Diagrams as examples (OMG, 2005). Each business process model contains several business process steps (BPS) which together describe a set of activities. Additionally, inputs and outputs of the process steps – i.e. information objects (IO) – are commonly contained in a process model. Finally, process steps are usually assigned to one or more actors (e.g. roles, departments, computer systems, etc.). 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*. 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 architect (e.g. legacy structures or different projects). In contrast to process steps (functionality) and objects (resources), which have a direct equivalent in 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 the single process steps is determined through the control flow. These dependencies are mapped onto edges between the corresponding BPS nodes (edge type *BPS-BPS*). Additionally, a process model describes the data flow between process steps, such that each step can have input and output information objects. These relationships are mapped onto edges between BPS and IO nodes (*BPS-IO*). Finally, entity relationships between information objects in a data model are mapped onto corresponding edges as well (*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; as well as related-to and state-of for IO-IO relationships. As mentioned before, additional edge and node (sub-) types could be used as well, if necessary. Thus, a graph-based approach is highly flexible with respect to the architect's and project's demands.

To assist the architect and support a rational decision upon the utilized weights, decision support methods are required (Zhu et al., 2005). Using pairwise comparisons of different edge (sub-) types (Figure 2), weights can be systematically derived in an Analytic Hierarchy Process (AHP, Saaty, 1980). Hence, the AHP supports a reflective determination of weights through reducing the decision space.

Compared to (A) creating information objects in business process steps, how important is (B) reading information objects in business process steps? Mark the relative importance on the AHP-scale:

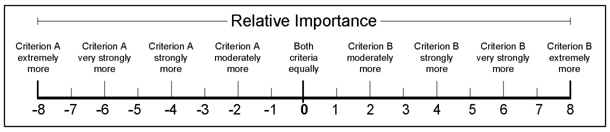

Relative Importance
Criterion A extremely more
Criterion A very strongly more
Criterion A strongly more
Criterion A moderately more
Both criteria equally
Criterion B moderately more
Criterion B strongly more
Criterion B very strongly more
Criterion B extremely more
-8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8

Figure 2. Pairwise comparison in an AHP

Pairwise comparisons have to be evaluated by the architect for all types of edges. Furthermore, for each general edge type the AHP has to be repeated on a second level for all corresponding subtypes. The comparison results are summarized in a matrix. Afterwards, impact factors for each type and subtype can be derived through eigenvalue calculations (Saaty, 1980). Based on the impact factors of types and subtypes, the exact weights of all edges are derived as product of both. Additionally, it has to be verified that no contradictory choices have been made by the architect. The consistency ratio CR of the matrix is commonly calculated and used as a first indicator for such problems. 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 (Saaty, 1980). Otherwise, the architect would have to reevaluate the choices made.

4.2 Optimized Partitioning

The derivation of an optimally structured IS landscape is not an easy task as a large 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 through comparing all combinations is unreasonable for non-trivial problems. Nevertheless, a combination of opening and improving heuristics can be used to identify the best possible solution with suitable effort. 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. For the partitioning we rely on the approach of Albani et al. (2008). Among the ones discussed in section 2 it turned out to be one of the most mature approaches (Albani et al., 2008, Birkmeier and Overhage, 2009). It uses the *Start Partition Greedy* (SPG) heuristic to generate the initial solution and the *KL-algorithm* from Kernighan and Lin (1970) for the subsequent improvement. The SPG heuristic aims to assemble closely related nodes in 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 business components. A specific strength of the method is that the architect does not have to decide on the number of components in the beginning, as it is determined by the heuristic.

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 structuring goal (i.e. a single large or many small one-step business components) can be avoided. For each solution, the algorithm calculates the coupling and cohesion ratios of the component structure.

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

$$(\mathbf{6}) \quad \overline{Cou(P)} = \frac{Cou(P)}{\sum_{\forall u,v} w_{(u.v)}} \in [0,1]; \qquad (\mathbf{7}) \quad \overline{Coh(P)} = \frac{Coh(P)}{\sum_{\forall u,v} w_{(u.v)}} \in [0,1]$$

Ratio (6) can be used to measure the target achievement of the minimizing coupling principle. Lower values are generally favorable here, as this indicates a lower amount of relationships between the business components. The second ratio (7) evaluates the maximizing cohesion target, accordingly.

4.3 Validation and Manual Improvement

As discussed earlier, the results of the automated derivation of business components should not be used without reflection. Especially, 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 optimized solutions are calculated for every set of weights. In previous applications of our approach a variation of ±50% of each impact factor in steps of ten percentage points has shown to be sufficient. The 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 et al., 1998). Here, 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 redistributed allows conclusions on the robustness of the component architecture.

In our approach we support the architect in examining node distributions over different optimization results through visualization in parallel coordinate plots. In such a plot, the axes are drawn parallel 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 (Inselberg, 1985). This method is frequently used in statistical data analysis for “examining clustering in high dimensions” (Cook and Swayne, 2007). Figure 3 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 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.

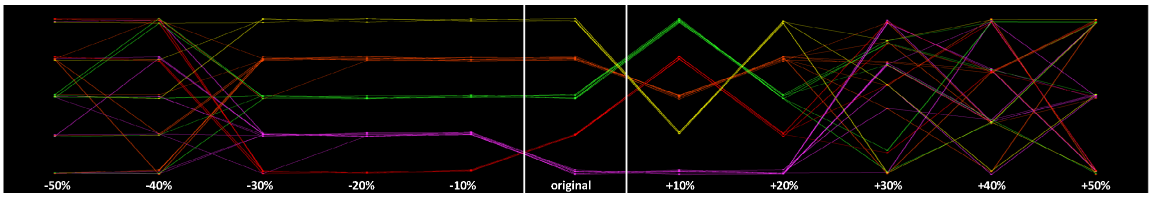


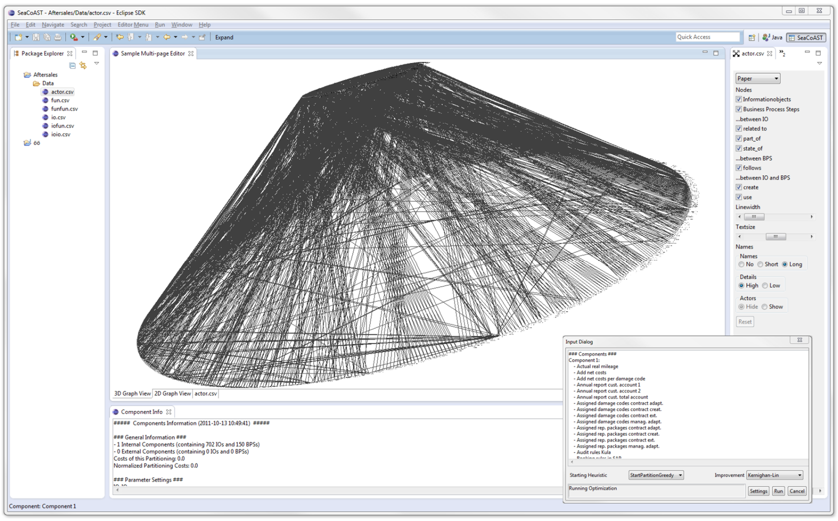
Figure 3. Parallel coordinate plot in sensitivity analysis

An examination for grouped lines reveals whether nodes keep together independently from the utilized weights. In the left part of Figure 3 most groups of nodes stick together up to -30%, which shows that the results are rather robust towards a 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 towards the component structure (c.f. bottom lines at *original*-axis and *-10%*-axis). On the right side of Figure 3 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 architecture. As the robustness of a component-based EA 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 architect 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 architect also might use the structuring as a profound starting point for further refinement through manual manipulations to give consideration to information that were not mapped onto the graph (e.g. financial aspects). However, since results from the optimization could be overruled and changes might cause the building of business components with lower quality regarding coupling and cohesion, we recommend this step to highly experienced architects only. Finally, the architect has to examine the business functionality which each component performs and initiate the business component specification.

5 Tool Support and Application

For large problems, the described method cannot be applied by hand but has to be supported through software. To furthermore validate the general applicability of our approach, we have developed the ***S**ervices **a**nd **Co**mponents **A**rchitecture **S**upport **T**ool* (SeaCoAST). It implements the described method and 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 applications up to large IS landscape projects. The following describes its application to the after-sales processes of a DAX-30 automobile manufacturer. All data has been derived in a project with our industry partner (Eberhardt, 2005). The business processes contain 150 steps that are carried out from 21 actors, affecting 702 information objects.



```
#####  Components Information (2011-10-13 10:49:41)  #####

### 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
Normalized Partitioning Costs: 0.0

### Parameter Settings ###
```

Input Dialog

```
### Components ###
Component 1:
  - 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 Kula
```

Starting Heuristic: StartPartitionGreedy
Improvement: Kernighan-Lin
Running Optimization
Settings Run Cancel

Figure 4. SeaCoAST with a visualization of the initial graph and optimization dialogs

The initial structure of the graph – as it is visualized in SeaCoAST – is shown in Figure 4 together with the optimization and relationships info dialogs. The top circle of nodes 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 model in this particular scenario are of the following subtypes. Within a process one step *follows* another and each one is assigned to an *actor*. All process steps can *create* information objects, or *use* them during their execution. Information objects can be *part of* larger objects, represent a certain *state of* another object, or simply be *related towards* each other.

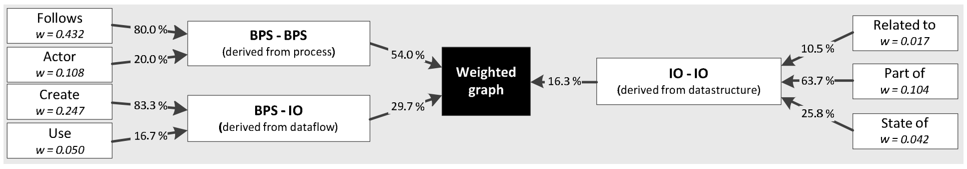


Figure 5. Weights w and impact factors derived through AHP

After mapping the information from the project's conceptual models (EPC and ER Diagrams) onto the graph, the optimization preferences had to be examined. Weights were derived through three experts based on an application of the AHP method. Among the experts involved in the discussion and voting, two were familiar with the project and the conceptual models. The third expert was acquired to provide an external view without knowledge of the specific project details. The resulting weights after eight pairwise comparisons are depicted in Figure 5. They reflect that among all edges, a special focus is placed onto those running between process steps. In the project, functional relationships are considered the most important to achieve reasonable business components, as large network traffic might present a problem. In detail, *follow* relationships *(w=.432)* are regarded more crucial than the influence of *actors (w=.108)*. Furthermore, information objects are to be placed preferably in the same component as the process step which *creates (w=.247)* them. The *usage (w=.050)* of data in process steps, however, is considered less complex and low weights are assigned to those relationships. The structure of information objects is evaluated as less important than functional relationships. Entity relationships are further ordered according to the individual subtypes: *part-of (w=.104)* is a stronger relation than *state-of (w=.042)* and is considered more important than *related-to* connections *(w=.017)*. The consistency ratio CR of the results varies between .000 and .033 with an overall value of .010. All of these are far below the critical value of 0.1 and therefore confirm that the derived weights contain negligible inconsistencies.

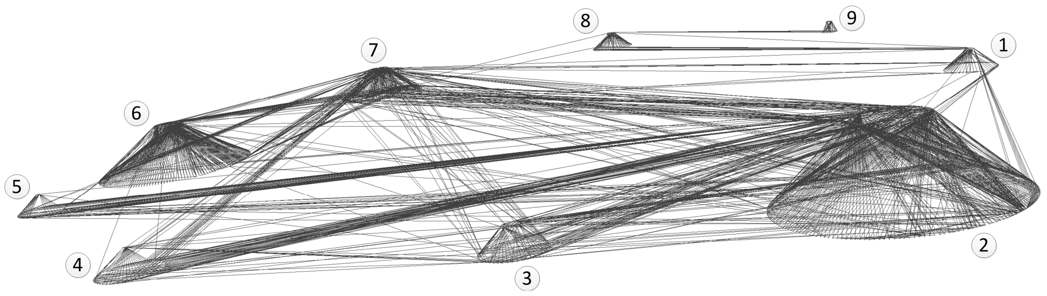


Figure 6. Visualization of the result after Start Partition Greedy and Kernighan-Lin heuristics

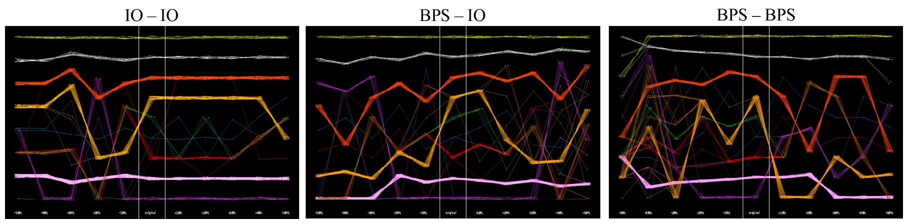
After running the SPG and KL heuristics, the graph is partitioned into nine components (Figure 6). The respective costs are reduced to $C(P) = 5'722$ with $\overline{Cou(P)} \approx 0.100$ and $\overline{Coh(P)} \approx 0.900$. All identified components were carefully analyzed and interpreted by the architects from a business perspective and described according to their main tasks (Table 2).

	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

Table 2. Summary of identified components

The architects performed an extensive sensitivity analysis with structured variations of the impact factors. Figure 7 visualizes the effect of changes in the range of ±50% to the original values for IO-IO, BPS-IO and

BPS-BPS relationships (Figure 5). Detailed examinations on 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 of 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 solutions when breaking points 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 weights of BPS-BPS relationships any more. Similarly, for a variation of weights in BPS-BPS relationships of more than -30%, the structure changes, as those relationship types now have less relative importance than BPS-IO relationships.



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%]

Figure 7. 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 can be used without limitations. From the project's perspective, the sensitivity analysis confirmed that the derived components are specific to the project's preferences and only a substantial change in goals results in a different IS design. In that context, the method itself and the results from the application of the introduced method have been corroborated by our industry partners.

6 Conclusions

In this paper, we have presented a method to systematically derive business components from conceptual models. In particular, we described the structuring strategy of minimizing coupling and maximizing cohesion as a multi-criteria decision-making problem. The presented method thereby allows the architect to decide which elements of conceptual models and which relationships between them should be considered during the structuring. Furthermore, it assists in identifying design preferences using a rational procedure. Finally, the presented method supports evaluating the stability of the resulting structuring against changing preferences. This evaluation allows the architect to identify and process emerging conflicts that may lead to a different structuring. To demonstrate how the method is used in practice, we introduced the SeaCoAST tool and discussed its application in a complex industry case. Project partners attested the method to provide an "interesting opportunity to balance technically optimal IS design with individual project requirements" while "supporting the architect but not suppressing him".

The results of our research have implications for both practice and academia. For practice, they show how the structuring of IS landscapes can be achieved using a rational procedure that can be customized according to situational requirements. In contrast to other methods, architects can adjust the structuring procedure to resemble their assumptions about the criteria that should drive the structuring. Possible clashes between the Weltanschauungs of the method creator and the method user, which lead to undesirable results when using methods (Omland, 2009), can so be minimized. The sensitivity analysis of the obtained results furthermore helps verifying the existing design preferences. Depending on the robustness of the created structuring, architects ought to rethink their preferences as the structuring 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 EA projects, where a considerable amount of components is anticipated. In such projects, the application of the presented method will likely lead to a structuring that is better optimized for the autonomy of its constituent components than manual approaches.

For academia, our results show how the derivation of business components from conceptual models can be achieved in a way that ensures active participation and influence of the architect while the structuring is still optimized systematically. The presented method provides a compromise in the current spectrum of fully automated methods, which do not allow for any influence or control, and manual approaches, which do not guarantee for a sufficiently optimized structuring. 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 structure a design space into business components. In contrast to existing approaches, the presented method provides support for a reflected, customized structuring. 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 acceptance. However, an extensive examination of the actual acceptance and possible efficiency gains yet has to be performed. Furthermore, there are also limitations towards 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. The method requires 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. We will therefore have to more rigorously evaluate the claimed advantages of our "reflective" method in future research. Besides further evaluating the presented method in field studies with industry partners, future research will also focus on determining exemplary weightings that were successfully applied in projects. So-called best practices can serve as benchmark for architects to analyze their own design preferences against. Finally, we will extend our method to the closely related service-oriented computing discipline. In that context, we currently work on systematically deriving business services from conceptual models to specifically support service-oriented enterprise architectures.

References

Albani, A., S. Overhage and D. Birkmeier (2008). Towards a Systematic Method for Identifying Business Components. In Proc. 11th Int. Symp Component-Based Software Engineering.

Birkmeier, D.Q. and S. Overhage (2009). On Component Identification Approaches - Classification, State of the Art, and Comparison. In Proc. 12th Int. Symp. Component-Based Software Engineering.

Blois, A., C. Werner and K. Becker (2005). Towards a Components Grouping Technique within a Domain Engineering Process. In Proc. 31st Conf. on Softw. Engineering and Advanced Applications.

Brown, A.W. (2000). Large-Scale, Component-Based Development. Prentice Hall, Upper Saddle River.

Cai, Z.G., Y.H. Yang, X.Y. Wang and A.J. Kavs (2011). A fuzzy formal concept analysis based approach for business component identification. Journal of Zhejiang University - Science C, 12 (9), 707-720.

Chidamber, S.R., D.P. Darcy and C.F. Kemerer (1998). Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. IEEE Transactions on Software Engineering, 24 (8), 629-639.

Cook, D. and D.F. Swayne (2007). Interactive and Dynamic Graphics for Data Analysis - With R and GGobi. Springer, New York.

Cui, J.F. and H.S. Chae (2011). Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. Information and Software Technology, 53 (6), 601-614.

D'Souza, D.F. and A.C. Wills (1999). Objects, Components, and Frameworks with UML: The Catalysis Approach. 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.

Fitzgerald, B., N.L. Russo and E. Stolterman (2002). Information Systems Development: Methods in Action. McGraw-Hill, London.

Ganesan, R. and S. Sengupta (2001). O2BC: a Technique for the Design of Component-Based Applications. In Proc. 39th Int. Conf. Technology of OO Languages and Systems, pp. 46-55.

Garey, M.R., D.S. Johnson and L. Stockmeyer (1974). Some Simplified NP-complete Problems. In Proc. 6th ACM Symposium on Theory of computing, pp. 47-63.

Herzum, P. and O. Sims (2000). Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise. John Wiley & Sons, New York.

Hevner, A.R., S.T. March, J. Park and S. Ram (2004). Design Science in Information Systems Research. MIS Quarterly, 28 (1), 75-105.

IBM Corporation (1984). Business Systems Planning: Information Systems Planning Guide.

Iivari, J. (2007). A Paradigmatic Analysis of Information Systems As a Design Science. Scandinavian Journal of Information Systems, 19 (2), 39-64.

Inselberg, A. (1985). The Plane with Parallel Coordinates. The Visual Computer, 1 (2), 69-91.

Jain, H., N. Chalimeda, N. Ivaturi and B. Reddy (2001). Business Component Identification - A Formal Approach. In Proc. 5th Int. Enterprise Distributed Object Computing Conf., pp. 183-187.

Jang, Y.J., E.Y. Kim and K.W. Lee (2003). Object-Oriented Component Identification Method Using the Affinity Analysis Technique. In Proc. 9th Int. Conf. on Object-Oriented Information Systems.

Kernighan, B.W. and S. Lin (1970). An Efficient Heuristic Procedure for Partitioning Graphs. The Bell Systems Technical Journal 49, 291-307.

Kim, S.D. and S.H. Chang (2004). A Systematic Method to Identify Software Components. In Proc. 11th Asia-Pacific Software Engineering Conference, pp. 538-545.

Lee, J.K., S.J. Jung, S.D. Kim, W.H. Jang and D.H. Ham (2001). Component Identification Method with Coupling and Cohesion. In Proc. 8th Asia-Pacific Software Engineering Conf., pp. 79-86.

Lee, S.D., Y.J. Yang, E.S. Cho, S.D. Kim and S.Y. Rhew (1999). COMO: A UML-Based Component Development Methodology. In Proc. 6th Asia Pacific Software Engineering Conf., pp. 54-61.

Levi, K. and A. Arsanjani (2002). A Goal-Driven Approach to Enterprise Component Identification and Specification. Communications of the ACM, 45 (10), 45-52.

March, S.T. and G.F. Smith (1995). Design and natural science research on information technology. Decision Support Systems, 15 (4), 251-266.

Mathiassen, L. (1998). Reflective systems development. Scandinavian Journal of Information Systems, 10 (1&2), 67-118.

McDonald, M., J. Begin and S. Fortino (2009). Meeting the Challenge: The 2009 CIO Agenda. Gartner.

Meng, F.C., D.C. Zhan and X.F. Xu (2005). Business Component Identification of Enterprise Information System: A Hierarchical Clustering Method. In Proc. IEEE Int. Conf. on e-Business Engineering.

OMG (2005). Unified Modeling Language Specification: Version 2. Revised Final Adopted Specification ptc/05-07-04, Object Management Group.

Omland, H.O. (2009). The relationships between competence, methods, and practice in information systems development. Scandinavian Journal of Information Systems, 21 (2), 3-26.

Ouyang, C., M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede and J. Mendling (2009). From Business Process Models to Process-Oriented Software Systems. ACM Transactions on Software Engineering and Methodology, 19 (2), 2-37.

Pannell, D.J. (1997). Sensitivity Analysis of Normative Economic Models: Theoretical Framework and Practical Strategies. Agricultural Economics, 16, 139-152.

Parnas, D.L. (1972). On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, 15 (12), 1053-1058.

Rodrigues, N.F. and L.S. Barbosa (2006). Component Identification Through Program Slicing. Electronic Notes in Theoretical Computer Science, 160, 291-304.

Saaty, T.L. (1980). The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation. McGraw-Hill, New York.

Sharp, J. and S. Ryan (2005). A Review of Component-Based Software Development. In Proc. 26th ICIS.

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

Sugumaran, V., M. Tanniru and V.C. Storey (1999). Identifying Software Components from Process Requirements Using Domain Model and Object Libraries. In Proc. 20th ICIS.

Szyperski, C., D. Gruntz and S. Murer (2002). Component Software - Beyond Object-Oriented Programming. ACM Press, New York.

Takeda, H., P. Veerkamp, T. Tomiyama and H. Yoshikawa (1990). Modeling Design Processes. AI Magazine, 11 (4), 37-48.

Vitharana, P. (2003). Risks and Challenges of Component-Based Software Development. Communications of the ACM, 46 (8), 67-72.

Vitharana, P., H. Jain and F.M. Zahedi (2004). Strategy-Based Design of Reusable Business Components. IEEE Transactions on Systems, Man and Cybernetics, 34 (4), 460-474.

Zhu, L., A. Aurum, I. Gorton and R. Jeffery (2005). Tradeoff and Sensitivity Analysis in Software Architecture Evaluation Using Analytic Hierarchy Process. Software Quality Journal, 13, 357-375.