Towards a Systematic Method for Identifying Business Components

Antonia Albani[1], Sven Overhage[2], and Dominik Birkmeier[2]

[1] Information Systems Design,
Delft University of Technology,
Mekelweg 4, 2628 CD Delft, The Netherlands
a.albani@tudelft.nl
[2] Component and Service Engineering Group,
Business Informatics and Systems Engineering Chair,
University of Augsburg,
Universitaetsstrasse 16, 86159 Augsburg, Germany
{sven.overhage,dominik.birkmeier}@wiwi.uni-augsburg.de

Abstract. The identification of business components, which together define a *modular* systems architecture, is a key task in todays component-based development approaches for the business domain. This paper describes the Business Component Identification (BCI) method which supports a systematic partitioning of a problem domain into business components. The method allows the designer to state preferences for the partitioning process and uses them as the basis to produce an optimized balance between the business components' granularity on the one hand and their context dependencies on the other hand. It makes use of business domain models specified during the definition of system requirements and can be integrated into the early design phase of a component-based development process. The paper also shows how the produced partitioning can easily be refined into an architecture specification and thus can be used as a starting point for the technical design of a software system and/or its business components.

1 Motivation

Modern component-based approaches allow developers to realize software systems in business domains by partitioning a problem space into a set of proper business components, developing or discovering suitable candidates, and assembling them to obtain the aspired solution [1,2,3]. This modular way of systems development promises to bring many advantages, among which especially a reduced time to market, the increased adaptability of systems to changing requirements and, as a result, reduced development costs are of key importance for the IT strategy of todays enterprises [1,4].

A prerequisite for the envisioned breakthrough of component-based approaches in practice, however, is to better support the underlying modular development paradigm with specialized methods and tools. Although the modular paradigm sounds rather straight-forward at a first glimpse, it introduces a variety of methodological challenges when being analyzed more closely. As a consequence, both the partitioning as well as the composition process continue to pose research questions. Compared to the composition process, where a lot of research is ongoing and for which methods to browse, adapt, as well as to assemble components in a predictable way have already been proposed [5,6,7,8,9], especially the question of how to partition a problem space into modular components still remains to be addressed.

In line with this observation, component identification strategies found in literature are usually limited to basic guidelines or general advices. The established partitioning principle of *maximizing cohesion and minimizing dependencies* between components, e.g., states that contextually related functions and data should be grouped together and ideally constitute a single component [10,11,1]. This principle, however, does not make a statement about an optimal component granularity. Consequently, it might be conceivable to design coarse-grained components containing all required functionality and having no context dependencies at all. Because this leads to redundant implementations of supporting functions and makes components more difficult to maintain, an alternative is to outsource supporting functions into separate components and opt for a better reuse grade. In practice, designers will have to strive for an *optimal balance between self-containedness and implementation reuse* [1]. This means that even with the advices and guidelines from literature taken for granted, a component-based system can well be partitioned into parts of varying size and context dependencies. To date, there only exist generalized approaches that show how a grouping of functions can technically be realized (see e.g. [12]) and discussions about different aspects that have to be taken into account (e.g. selected aspects of scale and granularity presented in [1]). The partitioning itself is still left completely to the designer and his or her personal skills, though.

In this paper, we present the *Business Component Identification (BCI)* method, which systematically supports the partitioning process and helps designers to find an optimized set of business components. The presented method takes results from the requirements definition as input and forces designers to make their partitioning preferences explicit. Based on these preferences, it generates an optimized partitioning of a problem space into business components, which provides a basis for further refining. It allows designers to make use of a rational, unequivocal partitioning procedure and validate the stability of the result against modified preferences. In doing so, the BCI method contributes to evolve the partitioning of component-based systems from handcrafting to an engineering process. The key research questions addressed in this paper are a) how the information modeled in business domains can be used to identify business components in a formal way and b) which optimization methods are suitable for the identification of business components leading to better results than existing solutions. Principally, the introduced partitioning algorithm is not limited to business domains, since it uses process and concept models as inputs which are being created in many application domains. To date, however, we have only applied BCI in business domains.

In section 2, we firstly discuss how to integrate BCI into the component-based development process. This discussion will also elaborate on the input that can be taken as a basis for the partitioning as well as the output that has to be generated by the BCI method. In section 3, we will then describe the BCI method in detail and present the algorithms used for the generation as well as the optimization of a partitioning. Section 4 briefly presents related approaches. We conclude the paper with a discussion of additional aspects that will be taken into consideration in the future and the work that has been done to validate the results of BCI in practice.

2 Systems Development and Component Identification Process

The partitioning of a problem space into components is a core part of the component-based development process and has a significant impact on the quality of both the resulting software system as well as its constituent components. Component-based development process models presented in literature therefore typically either comprise an explicit component identification phase before the actual design starts or at least include this task as an early step of the system design phase [2,13,12]. The extent, to which a partitioning has to be made from scratch, of course, depends on the availability of components that eventually can be reused.

With mature component markets in place and components preferably being reused instead of being newly developed, the partitioning process during the design of a software system needs to be driven by two determinants: the *predefined architecture* imposed by reusing existing components and the *conceptual models* created during the requirements definition. The conceptual models describe processes and information of the problem domain which have to be managed. There are various process models that can be used to develop component-based systems *with reuse*, among which the *Reuse-Oriented and Reuse-Driven Development* approaches [2] as well as the *Assemble Route* of Catalysis [13] are the more prominent ones. In such a reuse-oriented scenario, the partitioning of a problem domain into components also is an important step during the so-called development *for reuse*, which provides reusable components for the development of systems. Reusable components are usually not being developed in isolation, but in so-called domain engineering approaches in which entire problem spaces are being partitioned.

The before-mentioned reuse-based development has repeatedly been described as an ideal component-based software engineering scenario in literature. Using a component-based development approach, however, even is able to bring substantial benefits where component markets and in-house reuse are not established, since modular systems with easily replaceable parts better support managing changes [12,1]. Cheesman and Daniels have presented a process model that supports component-based systems development without a special focus on reuse [12]. In this case, the partitioning process can begin *from scratch*. It solely depends upon domain-oriented conceptual models that have been created during the requirements definition. Notably, however, is the fact, that none of the process models mentioned in this chapter describes how to achieve a good partitioning in detail. Instead, all of them are limited to giving very heuristic advice or to introducing technical means which merely help to capture relationships and dependencies between parts of the domain models. To advance the state of the art, we integrate a rational partitioning procedure, namely the BCI method, into the component-based development process.

The integration is demonstrated for the *UML Components* process model introduced by Cheesman and Daniels [12], which we have chosen for several reasons: firstly, mature component markets today are rather the exception than the rule and especially the development of business systems can not yet systematically include the reuse of existing components. Furthermore, the UML Components process model is well-established, easily applicable in practice, and – thanks to its close relationship to Catalysis as well as to other approaches [12, p. xv] – the transfer of our results to different process models is rather straightforward.

The UML Components process already includes an explicit component identification phase. It is part of the system specification and follows immediately after the requirements engineering (see fig. 1). The goal of the component identification phase is to come up with an initial specification of the system's architecture and its constituent components, which is then refined during the next design steps. The system partitioning is driven by the description of the problem domain and – following established software engineering principles – separates the discovery of system components (the front-end side) from the discovery of business components (the server side providing the business functions).

In this paper, we focus on the discovery of business components, which is based upon the *business concept model* and the associated *business processes*. The business concept model documents the information which is being processed in the application domain. It consists of *information objects* (concepts) and identified structural relationships between them. The business processes describe workflows of the business domain which have to be supported by the software system. They contain *business functions* (modeled as process steps) as well as the temporal relationships between them.

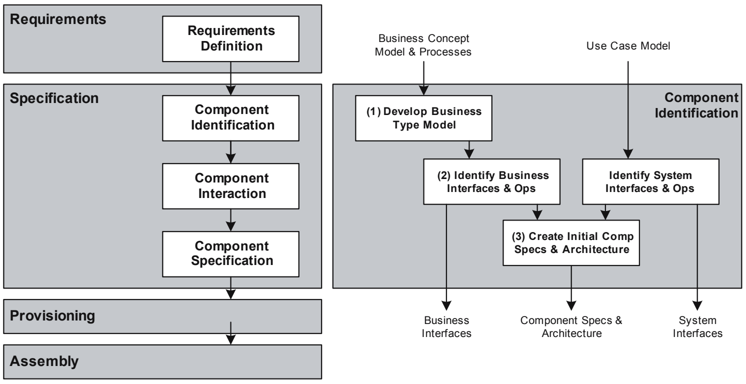


Fig. 1. UML Components process and component identification stage (cf. [12])

To identify business components, Cheesman and Daniels recommend to formalize the business concept model into a more detailed and technical *business type model* (see fig. 1 (1)). The next step is to identify so-called *core business types*, which represent information that can stand alone in the business domain. For each core business type, a business interface has then to be created (see fig. 1 (2)). A business interface has to manage the information represented by an independent core type and thus is a candidate to constitute a *business component*. The so identified business components finally have to be specified in detail and, together with identified system components, can be formed into an initial *systems architecture* (see fig. 1 (3)).

While this procedure may serve as a very heuristic approach to get to an initial set of business components, it has a variety of drawbacks. Firstly, even information that can stand alone in the business domain may likely have relationships to other information objects. Cheesman and Daniels acknowledge this and recommend to convert such

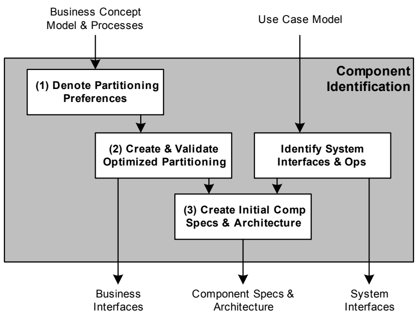

Business Concept Model & Processes
Use Case Model
Component Identification
(1) Denote Partitioning Preferences
(2) Create & Validate Optimized Partitioning
Identify System Interfaces & Ops
(3) Create Initial Comp Specs & Architecture
Business Interfaces
Component Specs & Architecture
System Interfaces

Fig. 2. Integration of the BCI method into the UML Components process

relationships into component dependencies. This might, however, not lead to an optimal partitioning, e.g., when components have to be easily replaceable and dependencies have to be kept at a minimum. Furthermore, the procedure is focused solely on a modularized management of information objects. Most business systems, however, will also have to include business components that manage entire processes or parts of business workflows [3]. These business components will then automatically have dependencies to all the business components governing relevant information in a process. Designers will hence have to take business functions *and* information (with their respective relationships to each other) into account when partitioning a problem space [10].

With the BCI method, the procedure proposed by Cheesman and Daniels can be replaced (see fig. 2). In line with their procedure, BCI also takes a business concept model and specified business processes as input to create a partitioning. In a first step, the designer will have to denote his partitioning preferences (see fig. 2 (1)). Thereafter, an optimized partitioning with respect to the given preferences is derived and has to be validated (see fig. 2 (2)). The resulting partitioning clusters process steps and information objects to form a set of business components. The identified business components are then to be refined, technically specified, and, together with the required system components, formed into a systems architecture (see fig. 2 (3)).

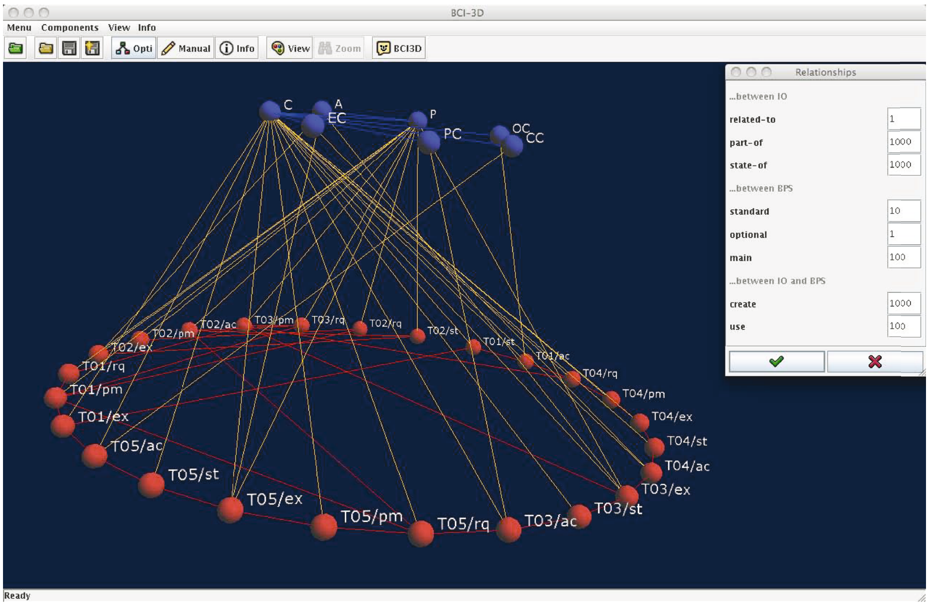
3 The Business Component Identification Method

The set of domain models and the defined metrics of maximizing cohesion and minimizing dependencies constitute the basis for the identification of business components. The identification is strongly dependent on the underlying domain models [14,15]. Only a domain model reflecting the business in a concise, complete and comprehensive way can lead to an adequate component model and therefore to a corresponding application system. In this paper we will not discuss the advantages and disadvantages of domain modeling methodologies. Instead, we will show how the information modeled in business domains can be used to identify business components in a formal way using the BCI method. Data from the domain of *Strategic Supply Network Development (SSND)* is used in the figures below to better visualize the identification process. The example domain comes from the area of strategic purchasing, where networks of suppliers are analyzed and selected in order to define an adequate purchasing strategy. It is not our intention to explain the SSND example in this paper, we rather focus on the formal method for identifying business components using the data of the SSND example. For details about the SSND domain we refer to [16]. In the following, the single BCI process steps – (1) Denote Partitioning Preferences, (2) Create and Validate Optimized Partitioning, and (3) Create initial Component Specification and Architecture – introduced in fig. 2 will be described.

3.1 Denote Partitioning Preferences

The BCI method uses the information objects from the concept models and the process steps from the process diagrams of the business domain, including their relationships. One can distinguish between three types of relationships necessary for the identification of business components: the relationships between single process steps, the relationships between information objects, and the relationships between process steps and information objects. A relationship type distinguishes between subtypes expressing the significance of a relationship. E.g., a relationship between single process steps expresses – based on their cardinality constraints – how often a process step is executed and therefore how close two process steps are related to each other in that business domain. The relationships between information objects define how loosely or tightly the information objects are coupled, and the relationships between process steps and information objects define whether a corresponding information object is, e.g., used or created while executing the respective process step. All types of relationships are of great relevance and build the basis for the BCI method. In order to apply a formal method for the identification of business components, we map the domain models to a weighted graph. As the nodes represent information objects and process steps, the edges characterize the relationships between the nodes. Weights are used to define the different types and subtypes of relationships. They build the basis for assigning process steps and information objects to components. The mapping of information objects and process steps from the domain models to nodes in the weighted graph is straightforward. Whereas, the definition of the relationship subtypes and the assignment of weights to corresponding edges is heavily dependent on the importance of the relationships in the underlying domain models. Therefore, domain knowledge and know-how is required for this step and the designers need to denote their partitioning preferences by introducing relevant relationship subtypes and assigning weights to them. The relationship subtypes as well as the weights may therefore differ dependent on the domain models and the preferences specified by the designers.

The BCI-3D Tool was developed to support the application of the BCI method. Due to display reasons the weighted graph is visualized in a three-dimensional representation having the process steps and information objects arranged as nodes in circles. The nodes representing the information objects are shown on top of fig. 3, and the nodes representing the process steps are shown on the bottom of fig. 3. The edges representing the relationships between information objects connect the top nodes to each other, the ones representing the relationships between process steps connect the nodes on the



Relationships

...between IO	
related-to	1
part-of	1000
state-of	1000
...between BPS	
standard	10
optional	1
main	100
...between IO and BPS	
create	1000
use	100

Fig. 3. BCI – defined preferences

Table 1. Assignment of process step names to shortcuts

process steps name	shortcut	process steps name	shortcut
request offering	T01/rq	state exploration	T03/st
promise offering	T01/pm	accept exploration	T03/ac
produce offering	T01/ex	request evaluation	T04/rq
state offering	T01/st	promise evaluation	T04/pm
accept offering	T01/ac	produce evaluation	T04/ex
request engineering	T02/rq	state evaluation	T04/st
promise engineering	T02/pm	accept evaluation	T04/ac
produce BoM explosion	T02/ex	request conclusion	T05/rq
state engineering	T02/st	promise conclusion	T05/pm
accept engineering	T02/ac	produce concluded contract	T05/ex
request exploration	T03/rq	state conclusion	T05/st
promise exploration	T03/pm	accept conclusion	T05/ac
produce contract	T03/ex		

bottom and the edges representing the relationships between information objects and process steps connect the nodes on top with the nodes on the bottom, shown in fig. 3. The weights assigned to the relationship subtypes are listed in a separate window on the right of fig. 3. Shortcuts are used to describe the process steps and information objects. The full names can be found in table 1 and table 2.

Table 2. Assignment of information object names to shortcuts

information object name	**shortcut**
Product	P
Assembly	A
Contract	C
Evaluated Contract	EC
Potential Contract	PC
Concluded Contract	CC
Offered contract	OC

3.2 Create and Validate Optimized Partitioning

For the identification of business components, as implemented by the BCI method, the weighted graph needs to be partitioned by assigning information objects and process steps to single components. The grouping should satisfy the defined metrics of maximizing cohesion and minimizing dependencies and should take all domain information into account which has been mapped to the weighted graph .

The problem of partitioning a graph $G = (V, E)$, with vertices V and edges E, into subsets of nodes of a defined size is known to belong to the class of NP-complete problems [17]. A clustering of the given example with 32 nodes into, e.g., three components of approximately equal size can be achieved in over 10^{12} different ways. Therefore, a direct calculation of the best solution by comparing all combinations is unreasonable, but heuristics can be used to find a best possible solution in suitable time. BCI applies an opening heuristic first, that gives a starting partition, and enhances this partition with an improving heuristic.

In general, a better starting partition is more likely to lead to better optimization results. We achieved the best results with the *Start Partition Greedy* heuristic shown in fig. 4. This is a greedy graph partitioning algorithm. In each iteration the most promising step is taken [18, p. 127]. The Start Partition Greedy utilizes a priority queue (*PQ*) to order the edges $e \in E$, whereby a higher priority is assigned to higher weighted edges. In the case of edges having equal weights, the weights of all edges adjacent to the end nodes are added. This allows for a fine-grained ordering. Beginning with unmarked vertices $v \in V$, the heuristic sequentially takes the edges in the *PQ*, and examines their end nodes. In the case of two unmarked nodes, a new component is generated. Whereas, in the case of one unmarked node, it is added to the marked node's component. Finally, all remaining unmarked nodes are collected in a last new component.

An advantage of our opening heuristic is, that there is no need to define the number of components in advance. It is determined by the Start Partition Greedy algorithm, depending solely on the given domain models and based on priority ordering of the edges. The idea is to cluster nodes, that are highly connected to their neighbors into one and the same component. An evaluation of different starting heuristics on various models, emphasized this method as leading to the most promising starting solutions.

After obtaining a primary solution for the optimization problem, various heuristics can be used to further improve the component structure. In 1970, Kernighan and Lin developed an algorithm for the enhancement of a given clustering of a graph into

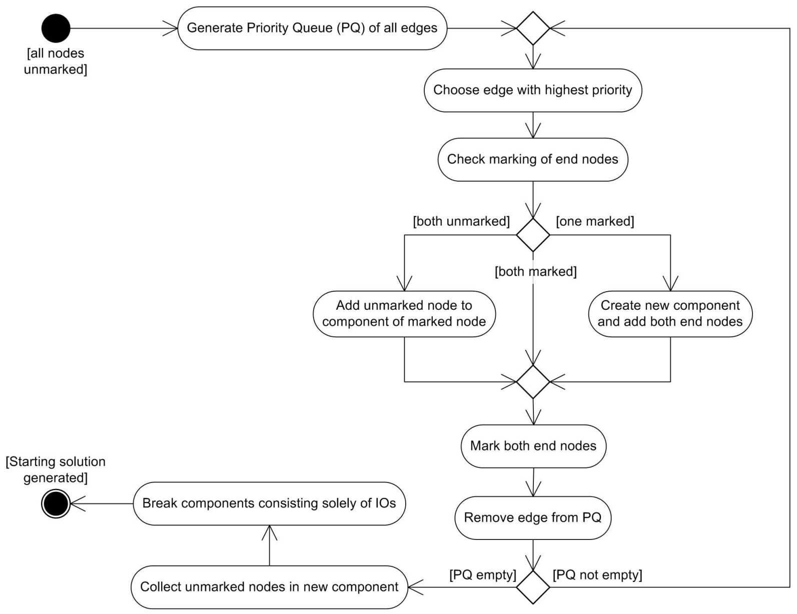

[all nodes unmarked]
Generate Priority Queue (PQ) of all edges
Choose edge with highest priority
Check marking of end nodes
[both unmarked]
[one marked]
[both marked]
Add unmarked node to component of marked node
Create new component and add both end nodes
Mark both end nodes
Remove edge from PQ
[PQ empty]
[PQ not empty]
Collect unmarked nodes in new component
Break components consisting solely of IOs
[Starting solution generated]

Fig. 4. UML Activity Diagram of the Start Partition Greedy heuristic

equal sized subgraphs [19]. Numerous variations of this method where proposed since then and all are based on the same concept (cf. [20,21,22]). We adopted the original Kernighan-Lin heuristics to improve the starting solution. This method does not consider all components at once, but rather a pair of two components in each step. At the beginning, all pairs are unmarked. In each step an unmarked pair is picked at random and the components are optimized, with respect to the heuristics. If any changes are made, all pairs are going to be unmarked again. Whereas, if no action is taken, the pair will be marked. This is repeated until no unmarked pairs are left and the component structure is optimized.

In order to compare different component structures and to be able to optimize them, we defined the cost $C(A, B)$ of a partitioning into components A and B as $C(A, B) = \sum_{a\in A, b\in B} w_{(a,b)}$, where $w_{(a,b)}$ is the weight of the connection between the single nodes $a \in A$ and $b \in B$. The goal is to minimize the cost of the partitioning for each pair of components (A, B). Furthermore, we defined internal $I(a)$ and external $E(a)$ costs of a node a according to Kernighan and Lin [19]:

$$I(a) = \sum_{x\in A, x\neq a} w_{(a,x)}, \quad E(a) = \sum_{b\in B} w_{(a,b)}$$

Moreover, the D-value of a node is referred to as the difference between its external and internal costs, $D(a) = E(a) - I(a)$. The gain $g(a, b)$ of exchanging the nodes a and b between the components A and B is then calculated by $g(a, b) = D(a)+D(b)-2w_{(a,b)}$. The basic procedure of a two-component optimization step corresponds to Kernighan-Lin and is shown in fig. 5.

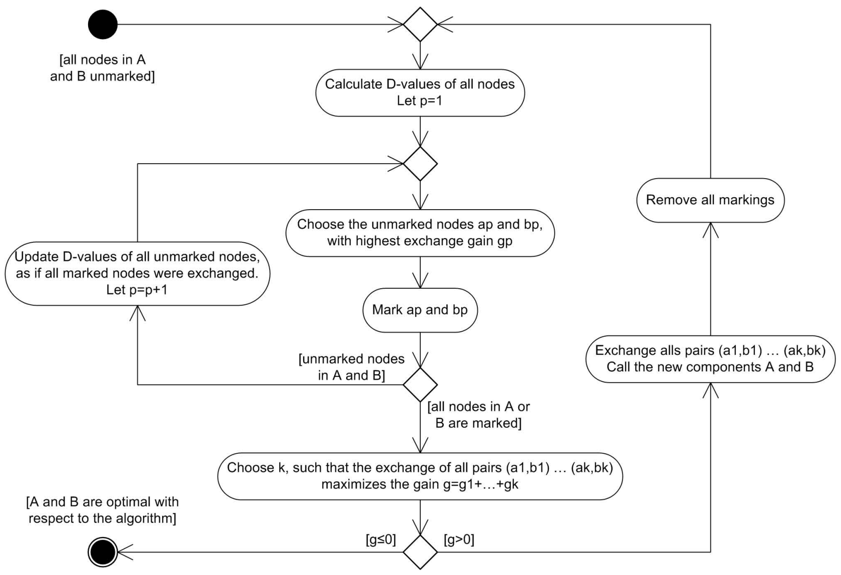

[all nodes in A and B unmarked]
Calculate D-values of all nodes Let p=1
Choose the unmarked nodes ap and bp, with highest exchange gain gp
Update D-values of all unmarked nodes, as if all marked nodes were exchanged. Let p=p+1
Mark ap and bp
[unmarked nodes in A and B]
[all nodes in A or B are marked]
Choose k, such that the exchange of all pairs (a1,b1) … (ak,bk) maximizes the gain g=g1+…+gk
[A and B are optimal with respect to the algorithm]
[g≤0]
[g>0]
Exchange alls pairs (a1,b1) … (ak,bk) Call the new components A and B
Remove all markings

Fig. 5. UML Activity Diagram of the adopted Kernighan-Lin algorithm

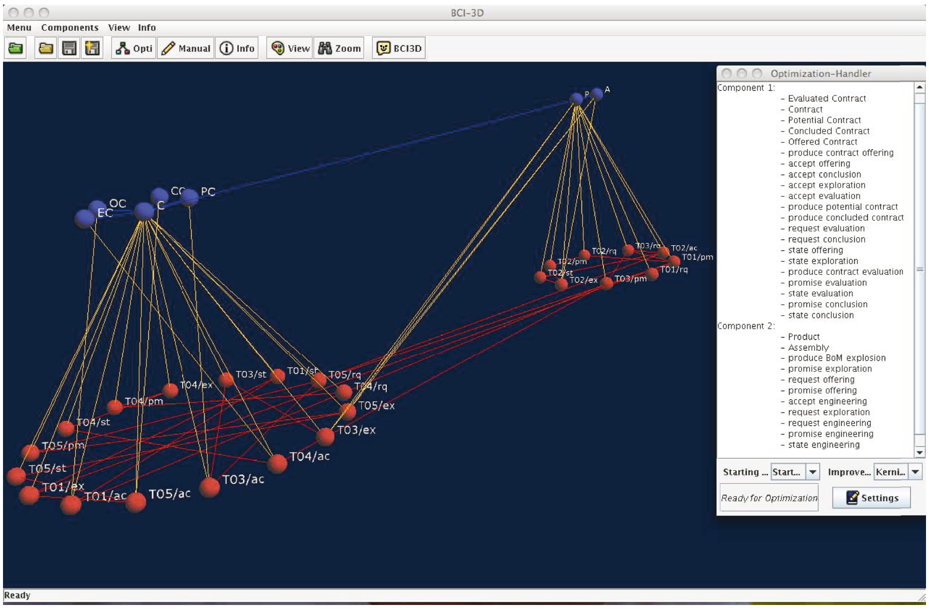
The process of identifying business components by applying the BCI method and satisfying defined metrics is an iterative process. The business components resulting from BCI need to undergo a sensitivity analysis check before taken for granted. In analyzing the process steps and information objects assigned to the resulting components, inconsistencies and errors in the underlying domain models can be identified and corrected correspondingly. Additionally, the resulting component model should remain stable even if the weights in the weighted graph are slightly changed. By changing weights of the relationships and reapplying the BCI method, the stability of the resulting component model can be analyzed.

Applying the BCI method to the graph introduced in fig. 3 results in the following graph clustering (see fig. 6). The figure shows the identified business components and the dependencies between them. Additionally, the window on the right lists the single process steps and information objects as assigned to the identified components by BCI.

3.3 Create Initial Component Specification and Architecture

Two business components can be identified immediately. While looking at the process steps and information objects clustered within the components, the designer can identify the business functionality of the two business components: one containing the business tasks related to *Product Management* and one containing the business tasks related to *Contract Management*.

From fig. 6, the services provided and required by each component can be derived. We distinguish between two types of services: *inter-component services* and *information services*. Inter-component services are services, which are required by another component in order to provide a specific functionality. The inter-component services


BCI-3D
Menu Components View Info
Opti Manual Info View Zoom BCI3D
P A
CC PC
OC EC C
T02/rq T03/rq T02/ac T01/pm T02/pm T02/st T02/ex T03/pm T01/rq
T03/st T01/st T05/rq T04/ex T04/rq T04/pm T05/ex T04/st T03/ex T05/pm T04/ac T05/st T03/ac T01/ex T01/ac T05/ac
Optimization-Handler
Starting ... Start... Improve... Kerni...
Ready for Optimization
Settings
Ready

Component 1:
- Evaluated Contract
- Contract
- Potential Contract
- Concluded Contract
- Offered Contract
- produce contract offering
- accept offering
- accept conclusion
- accept exploration
- accept evaluation
- produce potential contract
- produce concluded contract
- request evaluation
- request conclusion
- state offering
- state exploration
- produce contract evaluation
- promise evaluation
- state evaluation
- promise conclusion
- state conclusion

Component 2:
- Product
- Assembly
- produce BoM explosion
- promise exploration
- request offering
- promise offering
- accept engineering
- request exploration
- request engineering
- promise engineering
- state engineering

Fig. 6. BCI – optimized partitioning

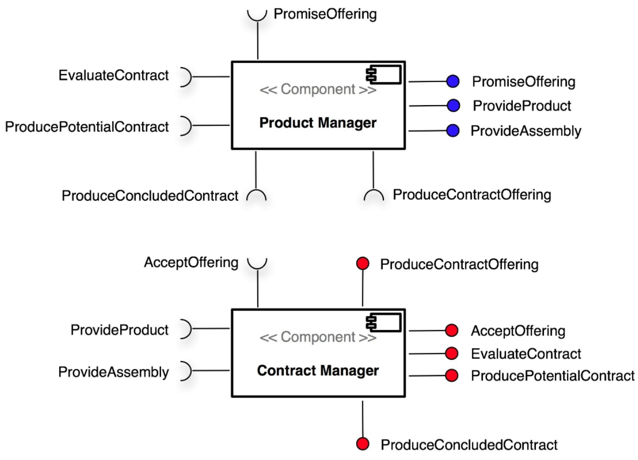


Fig. 7. UML Component Model of the identified components

are apparent in fig. 6 as the edges connecting two process steps, each located in a different component. E.g., the edge connecting the T01/pm (promise offering) and T05/rq (request conclusion) defines an inter-component service. The service provided by the Contract Manager component relates to the conclusion of the contract, and is therefore called ProduceConcludedContract. Which business component requires or provides that service becomes clear when looking at the process step diagrams of the relevant business domain. The identified business components with their required and provided services are shown in fig. 7.

The second type of services gained from the business components identified and visualized in fig. 6 are information services. While information objects are created and updated by the responsible business component, other components need to request the values of required information objects through services. By analyzing the edges that connect process steps of one component with information objects of another component the services are identified. In fig. 6 we have e.g., T03/ex (produce potential contract) connected by an edge with A (Assembly). This means that the process step of producing a potential contract needs information about the assembly information object. In this case the Product Manager component needs to provide the service ProvideAssembly, while the Contract Manager component requires that service (see fig. 7).

4 Related Work

The identification of business components and their services is a primary research problem that needs to be addressed. Today, there is still little research contributing to the development of systematic approaches which support designers in finding an optimized set of business components. In accordance with the classification introduced by [23], mainly three different types of business component identification techniques can be distinguished: *Domain Engineering* based methods, *CRUD (Create, Read, Update, Delete) matrix* based methods and *Cohesion-Coupling based Clustering Analysis* methods.

A key issue in the design phase of the domain engineering process is "the generation of components that represent conceptual, functional and technological aspects of the domain, and their organization within a domain architecture" [24]. Given that fact, Domain Engineering based methods for component identification usually focus on the reusability of the domain architecture and the adaptability of constituent components, based on defined criteria. E.g., the Feature-Oriented Reuse Method (FORM) [25] captures commonality selectable for different applications as an AND/OR graph, where AND nodes indicate mandatory features and OR nodes indicate alternative features. This graph is used to define parameterized reference architectures and reusable components instantiable during application development. Another approach aims at gathering components that intensively exchange messages in a unique artifact, and defining an architecture element referred to as components grouping [24]. It uses defined criteria for the grouping of components based on four different aspects: domain context, process component, components interfaces, and the component itself. Domain Engineering based methods, however, rarely use formal approaches to obtain reusable components and are highly dependent on the experiences of the designers.

CRUD matrix based methods focus on the semantics of business elements, which is contained in domain models, to merge closely related elements into business components. They use the relationships between behavioral business elements (e.g., process steps) and static business elements (e.g., information objects) to define how closely the elements are related to each other. Four relationship types – Create (C), Read (R), Update (U) and Delete (D) with the priority C>D>U>R – are used to specify the semantic relationship between the behavioral and the static business elements. The relationships are visualized in a matrix. CRUD matrix based methods aim at transforming the matrix by given rules in order to identify blocks in which behavioral and static business elements with C and D relationships are merged to form single components. Examples of CRUD matrix based methods are [26,27]. The disadvantage of CRUD matrix based approaches is that additional information available in the domain models is not used for identifying business components. E.g., the relationships between static elements and the relationships between behavioral elements are not considered at all.

With Cohesion-Coupling based Clustering Analysis methods, researchers try to cluster business models according to high cohesion and low coupling principles, and encapsulate each cluster in a component. The main idea of those methods is to first transform the domain models into the form of weighted graphs, in which business elements are nodes, the dependencies between single business elements are edges and semantic dependency strengths are represented as weights. In a second step, the graph is clustered using graph clustering or matrix analysis techniques that satisfy the metrics of high cohesion and low coupling. E.g., [28,29] are implementing such clustering analysis methods in order to identify components. Both approaches assume that UML models are available describing the business domain. The disadvantage of such approaches is that they are often based on technical concepts defined, e.g., in UML instead of focusing on the semantics of the corresponding business domain.

The BCI method directly contributes to the research area of identifying business components. According to the classification of business components identification methods by [23], the BCI method combines Cohesion-Coupling based Clustering Analysis and CRUD matrix based methods. The advantage of BCI is that the method uses all relevant dependencies of business domain models, including relationships between behavioral business elements, between static business elements, and those between behavioral and static business elements. It therefore extends CRUD matrix based methods with two additional types of dependencies. Additionally, BCI maps those business elements and their mentioned dependencies, independently of the notation used to model the business domain and its technical concepts, into a weighted graph. This graph is then used to apply the Cohesion-Coupling based Clustering Analysis methods implemented in BCI for identifying business components. With BCI, we thus satisfy Wang's recommendation of *combining* current component identification methods in order to achieve better results [23].

5 Conclusions and Future Directions

In this paper, we described the BCI method and have shown how to integrate it into the UML Components development process. The BCI method creates a partitioning of a problem space into business components which are optimized to satisfy the designers' partitioning preferences. In doing so, we advance the current state of the art and contribute to establish a more systematic approach to partition business systems into components, a key task in component-based systems development.

The BCI method was created in a perennial research project and has been continually improved to reach the scope of operation presented in this paper. It already has been *validated* in complex case studies that confirm its appropriateness for the development of component-based business systems in practice [30,31,32]. While the algorithms used in the current method and the derived partitioning results have proven to be mature, several approaches to further the scope of operation are currently under development. Among others, it is the plan to empirically evaluate the BCI method versus the other component identification methods described in this paper in order to show that the approach presented is superior to alternative approaches. Additionally, the initiative to integrate the BCI method into a tool that covers the domain modeling process is ongoing. With the partitioning as a final result, the tool may lead over to a component-based system design as well as to a service-based development approach. More technically motivated research initiatives include an automatic derivation of *component orchestrations* as well as the generation of parts of the components' internal structure.

In future, we plan to extend the BCI method to support *reuse-driven development approaches*, in which existing components will be considered. To reuse existing components during the partitioning process, we require conceptual models of process steps and information objects managed by those components. These models are either derived from technical specifications or already available when building upon more holistic specification approaches like, e.g., the Unified Specification of Components approach [33]. Existing components will then be represented as clusters of process steps and information objects that are marked to remain unchanged during the partitioning.

Our research initiatives centered around the BCI method are part of a longer-term goal to provide a mature methodical support of the partitioning process, just as it will become available for the complementary composition process. Only with an appropriate support of *both* processes, component-based development will lead to a component-based software engineering process.

References

1. Szyperski, C., Gruntz, D., Murer, S.: Component Software. Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley, Harlow (2002)
2. Sametinger, J.: Software Engineering with Reusable Components. Springer, Heidelberg (1997)
3. Herzum, P., Sims, O.: Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise. John Wiley & Sons, New York (2000)
4. Brown, A.W.: Large-Scale, Component-Based Development. Prentice Hall, Upper Saddle River (2000)
5. Zaremski, A.M., Wing, J.M.: Signature Matching: A Tool for Using Software Libraries. ACM Transactions on Software Engineering and Methodology 4(2), 146–170 (1995)
6. Seacord, R.C., Hissam, S.A., Wallnau, K.C.: Agora: A Search Engine for Software Components. Technical report CMU/SEI-98-TR-011, Software Engineering Institute, Carnegie Mellon University (1998)
7. Yellin, D., Strom, R.: Protocol Specifications and Component Adaptors. ACM Transactions on Programming Languages and Systems 19(2), 292–333 (1997)
8. Wallnau, K.C.: A Technology for Predictable Assembly from Certifiable Components. Technical Report CMU/SEI-2003-TR-009, Software Engineering Institue (2003)
9. Reussner, R.H., Schmidt, H.W.: Using Parameterised Contracts to Predict Properties of Component-Based Software Architectures. In: Crnkovic, I., Larsson, S., Stafford, J. (eds.) Workshop on Component-Based Software Engineering, Lund (2002)

10. Parnas, D.L.: On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM 15(12), 1053–1058 (1972)
11. Meyer, B.: Object-Oriented Software Construction, 2nd edn. Prentice Hall, Englewood Cliffs (1997)
12. Cheesman, J., Daniels, J.: UML Components. A Simple Process for Specifying Component-Based Software. Addison-Wesley, Upper Saddle River (2001)
13. D'Souza, D.F., Wills, A.C.: Objects, Components, and Frameworks with UML. The Catalysis Approach. Addison-Wesley, Upper Saddle River (1999)
14. Albani, A., Dietz, J.L.: The benefit of enterprise ontology in identifying business components. In: IFIP World Computing Conference, Santiago de Chile, Chile (2006)
15. Albani, A., Dietz, J.L., Zaha, J.M.: Identifying business components on the basis of an enterprise ontology. In: Konstantas, D., Bourrieres, J.P., Leonard, M., Boudjlida, N. (eds.) Interoperability of Enterprise Software and Applications, Geneva, Switzerland, pp. 335–347. Springer, Heidelberg (2005)
16. Albani, A., Müssigmann, N., Zaha, J.M.: A Reference Model for Strategic Supply Network Development. In: Reference Modeling for Business Systems Analysis, Idea Group Inc. (2006)
17. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete problems. In: STOC 1974: Proceedings of the sixth annual ACM symposium on Theory of computing, pp. 47–63. ACM, New York (1974)
18. Jungnickel, D.: Graphs, Networks and Algorithms, 3rd edn. Springer, Berlin (2007)
19. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell system technical journal 49, 291–307 (1970)
20. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: DAC 1982: Proceedings of the 19th conference on Design automation, Piscataway, NJ, USA, pp. 175–181. IEEE Press, Los Alamitos (1982)
21. Dutt, S.: New faster kernighan-lin-type graph-partitioning algorithms. In: ICCAD 1993: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design, pp. 370–377. IEEE Computer Society Press, Los Alamitos (1993)
22. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: Proceedings of the 1995 ACM/IEEE conference on Supercomputing. ACM, New York (1995)
23. Wang, Z., Xu, X., Zhan, D.: A survey of business component identification methods and related techniques. International Journal of Information Technology 2, 229–238 (2005)
24. Blois, A.P.T., Werner, C.M., Becker, K.: Towards a components grouping technique within a domain engineering process. In: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2005) (2005)
25. Kang, K.C., Kim, S., Lee, J., Kim, K., Kim, G.J., Shin, E.: Form: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5, 143–168 (1998)
26. Lee, S., Yand, Y.: Como: A uml-based component development methodology. In: Proceedings of the 6th Asia Pacific Software Engineering Conference, pp. 54–63 (1998)
27. Somjit, A., Dentcho, B.: Development of industrial information systems on the web using business components. Computer in Industry 50, 231–250 (2003)
28. Kim, S.D., Chang, S.H.: A systematic method to identify software components. In: 11th Asia-Pacific Software Engineering Conference (APSEC), pp. 538–545 (2004)
29. Jain, H., Chalimeda, N.: Business component identification - a formal approach. In: Proceedings of the Fifth International Enterprise Distributed Object Computing Conference (EDOC 2001). IEEE Computer Society, Los Alamitos (2001)
30. Albani, A., Bazijanec, B., Turowski, K., Winnewisser, C.: Component framework for strategic supply network development. In: Benczúr, A.A., Demetrovics, J., Gottlob, G. (eds.) ADBIS 2004. LNCS, vol. 3255, pp. 67–82. Springer, Heidelberg (2004)

31. Selk, B., Kloeckner, S., Bazijanec, B., Albani, A.: Experience report: Appropriateness of the bci-method for identifying business components in large-scale information systems. In: Turowski, K., Zaha, J.M. (eds.) Component-Oriented Enterprise Applications, Proceedings of the Conference on Component-Oriented Enterprise Applications (COEA 2005), Bonn, Köllen. Lecture Notes in Informatics, vol. 70, pp. 87–92 (2005)
32. Eberhardt, A., Gausmann, O., Albani, A.: Case study automating direct banking customer service processes with service oriented architecture. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 763–779. Springer, Heidelberg (2006)
33. Overhage, S.: UnSCom: A Standardized Framework for the Specification of Software Components. In: Weske, M., Liggesmeyer, P. (eds.) NODe 2004. LNCS, vol. 3263, pp. 169–184. Springer, Heidelberg (2004)