Towards a Components Grouping Technique within a Domain Engineering Process

Ana Paula T. B. Blois¹,², Cláudia M. L. Werner¹, Karin Becker²

¹ Federal University of Rio de Janeiro
COPPE/UFRJ – System Engineering and Computer Science Program
P.O. Box 68511 – ZIP 21945-970 – Rio de Janeiro – RJ – Brazil
² Catholic University of Rio Grande do Sul
PPGCC/PUCRS – Computer Science Graduate Program
Av. Ipiranga, 6681 – Prédio 30 – Bloco 4 – ZIP 90619-900 - Porto Alegre – RS - Brazil
{anablois, werner}@cos.ufrj.br {anapaula, kbecker}@inf.pucrs.br

Abstract

Many components are usually created within a Domain Engineering (DE) process. These components may exchange a great number of messages among them, and many times this interaction occurs within a specific group of components. The more messages the components exchange, the more complex the domain architecture becomes, thus compromising its understandability, maintainability and ultimately its reusability. This paper presents some elements of a components grouping technique that aims to identify and propose solutions to components coupling problems. This technique is being proposed within a domain engineering process called CBD-Arch-DE focused on component-based architectural design.

1. Introduction

Domain Engineering (DE) addresses the systematic creation of domain conceptual models, architectures and implementation models, with an emphasis on reuse and product lines. The domain analysis phase addresses the discovery and exploitation of commonality and variability across related software systems for achieving successful software reuse. The domain design phase aims at specifying a generic architecture for the domain to be reused by specific domain applications. A domain architecture can be represented by an architectural style that is appropriate for a given product family [11]. While domain analysis is a relatively well-established discipline, for which various methods exist (e.g. FODA [7]), domain design has received much less attention.

A key issue in domain design is the generation of components that represent conceptual, functional and technological aspects of the domain, and their organization within a domain architecture. The amount of components directly influences the architecture generation process, yielding components which interact by exchanging messages through their required and provided interfaces. The more messages the components exchange, the more complex the architecture becomes, thus compromising its understandability, maintainability and ultimately its reusability.

To improve the expected quality of a domain architecture, particularly understandability and reusability, it is advisable to gather components that intensively exchange messages in a unique artifact, defining an architecture element referred to as components grouping. These components are packaged together within a new component (components grouping), which becomes available in the domain artifacts repository, together with the original components. The task of components grouping has to be performed according to the following goals: a) to understand why components heavily interact; b) to organize components in the domain architecture according to adequate criteria; c) to produce domain architectures that are easy to understand; d) to improve the reusability of domain architectures, during an application engineering process.

The literature on Component-Based Development (CBD) does not present effective solutions to structure components within domain architectures in general, nor components grouping in particular. Methods such as UML Components [3], Kobra [1] and Catalysis [5] present CBD solutions to the identification and specification of components, considering their different views (e.g. structural).

There are some proposals found in the technical literature to evaluate coupling [4] [9]. Lee et al. [9] propose a method to identify components based on
coupling, cohesion, inheritance, dependency, interfaces, granularity and the architecture of their classes. It uses a set of guidelines to find key-classes and use cases categorized by functions (i.e., key, secondary or optional) with the goal of obtaining the coupling between classes. Thus it is possible to obtain coupling metrics based on the components internal specification (i.e., classes). Cho et al. [4] propose a set of dynamic metrics that measure the number of messages exchanged between objects at run-time. While static metrics [9] are obtained at class level, dynamic metrics are related to the object level [4]. These works address coupling issues in a granularity that is too fine-grained for components grouping, and thus have to be complemented to improve the cohesion and coupling of components at architectural level.

In this work, we address the coupling of components, to improve the cohesion of elements in a domain architecture. For that purpose, we propose a set of criteria that helps to understand the nature of components interaction and evaluate their dependencies, with the aim of supporting the components grouping task in the context of a DE Process. Component coupling issues are addressed by considering the messages exchanged between components through their provided/required interfaces, as well as various domain engineering artifacts defined in the DE analysis that express relationships between components. Although these criteria have been proposed in a DE context, most of them can be used in any software design activity to help the engineer to group components.

The rest of this paper is organized as follows. Section 2 provides a brief overview of the DE and CBD areas, and presents the CBD-Arch-DE Process. Section 3 presents some of the elements of a Components Grouping Technique. Section 4 describes a case study in Cooperative Learning domain to illustrate the technique. Section 5 presents some final considerations and directions for future work.

2. Domain Engineering and Component-Based Development

Many DE methods (e.g., FORM [8], FODA [7]) present techniques to model DE artifacts that are reused by domain applications. DE aims at identifying variable and invariable aspects of the domain to promote reuse, and most DE methods are based on the use of features to represent the conceptual, functional and technological characteristics of the domain in an abstract level, mostly recognized as domain analysis.

Reuse is also a concern of the CBD area, where variable and invariable aspects of a component are available through its interfaces. Nevertheless, in practice, CBD methods (e.g. UML Components [3], Kobra [1] and Catalysis [5]) are targeted at the identification, specification and connection of components to compose a specific application, with much less focus on producing reusable components for a certain domain.

An alternative would be to combine CBD and DE techniques. Through DE, software engineers can make available to reusers the artifacts of different phases, such as use cases and features resulting from the analysis, as well as components, produced in the design phase, to build the domain components architecture.

Braga et al. [2] proposed a process – Odyssey-DE – that unifies the aspects of reuse and domain understanding, as provided by DE methods, and the representation of domain (in)varyants as components, provided by CBD methods. Nevertheless, the set of activities referring to the domain design phase was not exhaustively explored. CBD-Arch-DE (Component-Based Development – Architecture – Domain Engineering) is a process proposed with the goal of detailing the activities referring to architectural issues in design phase, hence extending Odyssey-DE process.

2.1. CBD-Arch-DE Process

The first two phases of CBD-Arch-DE process, namely Domain Planning (e.g., estimate resources, schedules) and Domain Analysis (e.g., identify contexts, features, business types and use cases), are analogous to Odyssey-DE. These phases are finished when the respective output artifacts are created by the Domain Engineer. The output elements of the Domain Planning phase are documents and schedules about the domain. The output elements of Domain Analysis phase are domain contexts, domain features model, technological features model, business types model and use case model.

The Domain Design phase is composed by two activities: components generation and architecture generation, as presented in Figure 1. The steps of the components generation activity are: 1) creation of business and technological (i.e., utility and infrastructure [13]) components, 2) specification of business and technological components, and 3) creation and specification of process components. In the first step, the business and technological components of the domain are created. Architectural styles can be used to create business components with their provided and required interfaces, using the business types identified at domain analysis, as
proposed by Teixeira et al. [10]. Technological components with their provided interfaces can be created based on technological features also obtained from domain analysis. In the second step, these components are internally specified by a set of classes, their attributes, methods and relationships. Design and architectural patterns can be used to structure the class model [6] at this level. In the third step, components are created based on use cases taken from domain analysis. For each use case, a component process is created with provided and required interfaces, together with an interaction diagram that shows the interaction of business components described within the use case.

### Design Activities of CBD-Arch-DE Process

The architecture generation activity is composed by three steps: 1) identification of architectural properties, 2) components grouping, and 3) architecture creation. The first step aims at the definition of the domain architecture characteristics (i.e., architectural style to be chosen, organization of the domain characteristics, criteria to be used, priority among criteria etc), which will influence the next two activities of the architecture generation. The components grouping step aims at organizing the large number of components created by the previous components generation activity, using components grouping criteria. Component grouping is the focus of this paper and it is detailed in Section 3. The last step is the creation of the domain architecture structured in terms of original and grouped components, guided by architectural properties to create the domain architecture. The components grouping and architecture creation steps have a high level of interaction. A domain architecture is developed in a iterative process and new components grouping can be suggested during this process.

The last phase of the CBD-Arch-DE is the Implementation, which has not been thoroughly addressed yet. Currently, our research group is investigating how to map domain components into a specific component technology (e.g. Enterprise Java Beans).

### Components Grouping

Components grouping aims at gathering components produced in the Components Generation activity into coarser-grained components, according to coupling criteria. This is a crucial step towards the definition of domain architectures that are easier to understand, and promote reuse. Components coupling and cohesion issues are considered according to the domain characteristics expressed in all artifacts produced in the previous analysis and design activities (e.g. features and use cases of the analysis phase; components of the design phase).

The components grouping technique intends to identify and propose solutions to coupling problems that can happen in a DE process context (i.e. CBD-Arch-DE). So far, the following coupling problems were identified: a) components that are mutually dependent, i.e., they provide and require interfaces from each other; b) components requiring many interfaces of a restricted number of components; c) components exchanging messages with each other and all these components are traced to the same domain context. A context is defined during the domain analysis phase and aims to categorize all domain artifacts based on the different support offered by the domain (e.g. Pedagogical and Administrative can be two contexts of a Cooperative Learning Domain). To identify components coupling problems, an initial set of six criteria is proposed. By considering the results of these criteria, the Domain Engineer can decide which components should be grouped.

To the best of our knowledge, a components grouping technique was never proposed before in the context of a DE process. As mentioned before, DE and CBD methods do not present techniques or guidelines to support this task. Our components grouping approach uses components and interfaces previously generated through CBD-Arch-DE process to support the components grouping based on some coupling criteria.

#### 3.1 Components Coupling Criteria

The Domain Engineer uses the following input elements to perform components grouping activities: 1) business and technological components with provided and required interfaces, and their respective component specifications; and 2) process components with required and provided interfaces. The output are: a) a refined domain components architecture, composed of both original (with regard to components generation activity) and grouped components and b) a component grouping model that reveals how the
components grouping are composed and how the internal components exchange messages with between them.

Each criterion uses a specific artifact as input element. A criterion evaluates the responsibility and dependency of each domain component and generates information that helps the Domain Engineer to decide whether a set of components should be grouped together or not. To generate more useful results, a combination of criteria is recommended, guided by the architectural properties previously identified (Section 2.1). All criteria can be used alone but, in most cases, their combined use is indicated to improve the components grouping decision making. The combination of criteria can either be a decision of the Domain Engineer, or indicated by the results obtained at the end of a criterion execution. The second situation occurs when a criterion alone does not provide enough information for suggesting any grouping. When the criteria are used in a combined fashion, the grouping suggestions of a given criterion do not discard the components to be evaluated in the following criteria. In this sense, if a components grouping is suggested by a criterion, these components can also be used in the evaluation of another criterion.

The criteria are:

a) Coupling by Domain Contexts (CDC)

**Issue:** How many and which domain contexts each component is traced in?

**Rationale:** During DE analysis, domain features are organized in different domain contexts, which represent groups of common functionalities of the domain (e.g. sub-domain) [2]. A given feature can belong to more than one context. These features are used to derive business types, which in turn are used to define components. According to CDC criterion, if two distinct components are traced into features that belong to the same context, the grouping of these components is suggested to the Domain Engineer. It is important to point out that during the application engineering process, the application engineer must identify the domain contexts that he aims to reuse in his specific application. Thus, to be more understandable and reusable, domain architectures should group as much as possible components that are traced to the same context, and discourage grouping components that are traced into different contexts.

b) Coupling by Domain Process Components (CDP)

**Issue:** How many and which business components are required by each process component?

**Rationale:** Process components require interfaces of business components to execute their services. This criterion advises to group process components with

the business components that provide their required interfaces. By grouping these components, the domain architecture would be simpler because related process and business characteristics are encapsulated in a unique domain artifact.

c) Coupling by Provided Interfaces (CPI)

**Issue:** How many and which interfaces are provided by each component?

**Rationale:** This criterion yields the degree of responsibility of each component in the architectural level. If a component provides many interfaces, the grouping with the components that require them is suggested.

d) Coupling by Required Interfaces (CRI)

**Issue:** How many and which interfaces are required by each component?

**Rationale:** This criterion obtains the dependency degree of a component through its interfaces. If a component requires many interfaces, it can be suggested to group it with the set of components of which it requires interfaces. In this sense, if two components require interfaces from each other, then their grouping is recommended. Conversely, if the component does not require a component for which it provides interfaces, the grouping is not suggested.

e) Coupling by Required Components (CRdC)

**Issue:** How many and which components are required by a component?

**Rationale:** This criterion obtains the dependency degree of each component, without considering its interfaces. If a component requires many other components, their grouping may be suggested. However, another way to check this grouping is by evaluating if the required interfaces of a given components also require its services (CRnC). If the components require the services of the former, then the grouping can be suggested.

f) Coupling by Requiring Components (CRnC)

**Issue:** How many and which components are requiring a certain component?

**Rationale:** The criterion identifies if a component has many components requiring its services, without considering its interfaces. If a component is requiring other components, their grouping is suggested.

The criteria above suggest the grouping of components based on four different aspects: domain context, process component, components interfaces and the component itself. CPI, CRI, CRdC and CRnC criteria use the same artifact to evaluate the components grouping, but according to different points of view. When a component provides only one interface, CPI produces a similar result to CRdC.
However, when components provide more than one interface, CPI and CRdC produce different results. The Domain Engineer must choose the appropriated criteria, based on the characteristics of the components at hand. If the Domain Engineer wants to evaluate the amount of messages among components, he should use CPI. On the other hand, when he wants to know if a component is required by another one, he should use CRdC.

As mentioned before, one or more criteria can be used by the Domain Engineer during the components grouping task. Furthermore, a certain criterion result can also indicate another criterion to be used (e.g., CPI commonly indicates the combined use of CRI) after its execution.

Currently, heuristics to help the Domain Engineer during the architecture generation activity are under definition. These heuristics include some guidelines, which involve: a) the choice and use of architectural style(s) to build the domain architecture; b) the analysis of domain artifacts variability, which should be considered during the domain architecture construction; and c) the grouping of components.

4. Case Study

To illustrate the criteria proposed, we discuss its use in the Cooperative Learning (CL) domain. Figure 2 shows an extract from the component architecture diagram produced for this domain. The notation UML 2.0 [12] is used to represent components and interfaces. Provided interfaces are represented by a ball and the required interfaces by a socket. The domain models and artifacts have been generated in a Reuse Environment based on Domain Models called Odyssey [2], which supports the CBD-Arch-DE process. The component architecture diagram displayed in Figure 2 is the result of the component generation activity of CBD-Arch-DE design phase. It presents business, process and technological components with their required and provided interfaces. Other CL domain models are not being presented in this paper due to space limitations.

4.1 Illustration of CDC Criterion

There are three domain contexts in the CL domain (not shown in Figure 2): Administrative, Technological and Methodological. Each component of Figure 2 is traced into one and only one of them. Thus, according to CDC criterion, components of the same context should be grouped, based on the assumption that components traced to the same context exchange more messages among each other. However, this information alone does not provide enough basis for taking a decision about the grouping. Course and Discipline, which are components that are traced to the same context (Administrative), exchange messages. On the other hand, EvaluationGraph, Chat, NewsGroup and Email are all traced to the technological context, but the components architecture model reveals that they do not exchange any message. In this case, the use of other criteria to evaluate the coupling of component interfaces is indicated to obtain the messages exchange between components (e.g. CPI and CDP).

4.2 Illustration of CDP Criterion

Figure 2 shows four process components that belong to a same context. CreateCourse and ManagerCourse process components require the ICourse interface provided by Course business component. ManageDiscipline and CreateDiscipline process components require the IDiscipline and
ICourse interfaces provided by Discipline and Course business components, respectively. CDP criterion suggests two possible components grouping. Course can be grouped with CreateCourse and ManagerCourse, generating a GroupCourse component. Likewise, Discipline can be grouped with CreateDiscipline and ManageDiscipline, generating a GroupDiscipline component. Considering the dependencies of the original components, the new GroupDiscipline requires the services of GroupCourse through the ICourse interface.

4.3 Illustration of CPI Criterion

Figure 2 shows nineteen domain components and most of them provide only one interface. There are only two components in this model that provide more than one interface. The Email infrastructure component provides two interfaces and Bibliography business component provides three interfaces. These components do not carry enough responsibilities to justify the groupings. Therefore, we can conclude that CPI criterion should not be applied when components do not provide many interfaces. In that case, CPI criterion provides more valuable results.

4.4 Illustration of CRI Criterion

Table 1 shows the results obtained by applying CRI criterion in components that require interfaces of other components. Course requires many services of other components, but the components that provide these interfaces do not require services of Course. Content component requires an interface of Discipline, and vice-versa. This result suggests the grouping of these two components. Although Course and Discipline are not grouped by CRI, they can be grouped based on CRnC criterion. When a given component is required by a single component, their grouping is also recommended. For instance, the grouping of AcademicSystem and Profile is recommended, whereas the grouping of the Student component with Profile component does not seem to be appropriate, because IStudent interface is required by both Course and Profile business components. ManageDiscipline, CreateDiscipline, CreateCourse, ManagerCourse, Course and Discipline can be grouped, as suggested by CDP and CRI criteria.

4.5 Illustration of CRdC Criterion

Table 2 shows the results obtained by applying CRdC criterion on components that require interfaces of other components. Course and Class components require many components, which in turn do not require Course and Class, respectively. Content requires Discipline and Bibliography, but only Discipline requires Content component. This result suggests grouping Discipline and Content components. CRdC and CPI provide similar results when components provide few interfaces.

4.6 Evaluation of CRnC Criterion

Table 3 shows the results obtained by applying CRnC criterion over components that are required by some other component(s). Course and Discipline are the most required components of the component model. Thus two groupings are suggested: a) Course with CreateCourse, ManagerCourse, ManageDiscipline and b) CreateDiscipline and Discipline components with Course, ManageDiscipline, CreateDiscipline and Content. ManageDiscipline and CreateDiscipline require the services of Course and Discipline. In this case, a grouping that includes Course and Discipline, and all the components that require them is suggested. It is also suggested to group components that require each other, which in turn are not required by any other. In this sense, Profile can be grouped with AcademicSystem, because AcademicSystem is not required by any other component, as shown in Figure2.
4.7 Components Grouping Results

Figure 3 shows the components grouping result based on the proposed criteria. The following components groupings were suggested by the criteria:

**CDC**: all infrastructure components;

**CDP**: 1) to group *Course* with *ManageCourse* and *CreateCourse*; 2) to group *Discipline* with *ManageDiscipline* and *CreateDiscipline*;

**CPI**: there was no suggestion for components grouping in this case study;

**CRI**: 1) to group *AcademicSystem* with *Profile*; 2) to group *Discipline* and *Content*;

**CRdC**: 1) to group *Discipline* and *Content*;

**CRnC**: 1) to group *Course*, *CreateCourse*, *ManagerCourse*, *ManageDiscipline*, *Discipline*, *CreateDiscipline* and *Content*.

### Table 3: CRnC Criterion Results

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Requiring Components</th>
<th>Requiring Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course</td>
<td>4</td>
<td><em>CreateCourse</em>, <em>ManagerCourse</em>, <em>CreateDiscipline</em></td>
</tr>
<tr>
<td>Discipline</td>
<td>4</td>
<td><em>CreateCourse</em>, <em>ManageDiscipline</em>, <em>CreateDiscipline</em>, <em>Content</em></td>
</tr>
<tr>
<td>Bibliography</td>
<td>2</td>
<td><em>Content</em>, <em>InstructionalMaterial</em></td>
</tr>
<tr>
<td>Content</td>
<td>2</td>
<td><em>Discipline</em>, <em>InstructionalMaterial</em></td>
</tr>
<tr>
<td>E-mail</td>
<td>2</td>
<td><em>Course</em>, <em>Class</em></td>
</tr>
<tr>
<td>NewsGroup</td>
<td>2</td>
<td><em>Course</em>, <em>Class</em></td>
</tr>
<tr>
<td>Student</td>
<td>2</td>
<td><em>Course</em>, <em>Profile</em></td>
</tr>
<tr>
<td>Activity</td>
<td>1</td>
<td><em>Class</em></td>
</tr>
<tr>
<td>Chat</td>
<td>1</td>
<td><em>Class</em></td>
</tr>
<tr>
<td>Class</td>
<td>1</td>
<td><em>Discipline</em></td>
</tr>
<tr>
<td>EvaluationGraph</td>
<td>1</td>
<td><em>Course</em></td>
</tr>
</tbody>
</table>

Table 3: CRnC Criterion Results

Note that *GroupCourseDiscipline* and *AcademicSystem_Profile* are grouped components in Figure 3. A new component is created with a grouping component stereotype, which provides two Port Elements (*PGroupCourseDiscipline* and *PAcademicSystem_Profile*), as proposed by UML 2.0 [12]. These Ports provide and require all interfaces of the grouped components that are provided/required externally. Other components that exchange messages with components encapsulated within the grouping must require its Ports.

With regard to the original architecture (Figure 2), the resulting component architecture of Figure 3 has fewer components. It should be noticed that the process is iterative, and the components grouping and architecture creation steps are entwined. Thus the resulting groupings are considered as input for the (re)application of the criteria, until the final architecture is defined. Thus, although unfinished, we consider that the new domain architecture of Figure 3 has improved in terms of understandability, maintainability and reusability with regard to the original one (Figure 2). The cohesion obtained by the criteria improved the reusability in an application engineering context (e.g. applications can reuse the *GroupCourseDiscipline* with all business and process components, and their respectively provided and required interfaces). The maintainability improved because all encapsulated components must be evaluated when any update is to be executed in a grouped component. Finally, the organization of components into groups improved the comprehension of the domain to support the architecture generation step (Figure 1).

5. Final Considerations

In this work, we presented some elements of a components grouping technique to identify and
propose solutions to coupling problems that can happen in a DE process context. This proposal is part of the activities of a process, aiming at building a component-based domain architecture.

DE and CBD methods do not propose guidelines nor mechanisms to support the designer in the grouping components task. Although the set of criteria proposed in this paper is very simple, their use in a DE process allows to better structure the domain architecture.

The criteria proposed in this paper can complement the coupling metrics proposed in [4] [9], because their coupling occur between fine-grained elements (classes) and, our criteria propose the coupling between coarse-grained elements (components) at architectural level.

Although these criteria can help the domain engineer activities, they only consider the coupling and cohesion of components to group them. In this sense, these criteria could be extended to also consider the semantic of the other domain artifacts to suggest components grouping. For example, features obtained during the domain analysis have variability properties and consistence rules that can be used to suggest some criteria to components grouping.

Currently, we are specifying a set of heuristics to guide the Domain Engineer during the component grouping decision-making.

A Grouping Criteria Tool has been created to support the proposed criteria. This tool is a plug-in of Odyssey Environment. The Domain Engineer chooses the criteria and the components over which to apply the criteria. The tool shows all available information of each criterion, suggestions of components groupings and related criteria, as necessary. Through the Grouping Criteria Tool we intend to improve the comprehension and organization of the domain components architecture, and ultimately, its reuse in applications.

A complete example in the CL domain is under development to evaluate the proposed technique. Up to this moment, the domain component architecture has around sixty components. This number can be effectively decreased by applying the technique. In this context, we intent to plan empirical studies to observe how well the approach helps to improve the CL domain architecture.

6. References


[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.


