2009 Sixth International Conference on Information Technology: New Generations

A Framework for Identifying Reusable Software Components Using Formal Concept Analysis

Haitham S. Hamza
Department of Information Technology
Faculty of Computers and Information
Cairo University
Egypt

Abstract

Component-based development (CBD) is concerned with enabling the design of software systems by reusing pre-developed components. One important activity in CBD is to analyze the business domain and identify potential reusable domain components. Identifying business components, however, can be challenging due to the different nature of the application requirements even within the same domain. Moreover, the rapid change in customer needs and business requirements might render domain components obsolete if these components can not accommodate changes with minimal cost and effort. This paper presents a novel framework for identifying and encapsulating stable domain components. The framework makes use of the theory of Formal Concept Analysis (FCA) and the concepts of Software Stability Model (SSM) to identify stable domain components. The details of the proposed framework are described and then demonstrated by the means of a case study.

Keywords: Component-Based Software Engineering, Software Stability, Formal Concept Analysis, Enduring Business Themes, Business Objects.

1. Introduction

The concept of software reuse can be traced back to the mid and late 1960s when software community has realized the need, as well as the challenge, for developing systematic software production tech*niques to improve productivity [1]. A promising approach to realize systematic reuse is through components. A component can be defined as "An independently deliverable unit of software that encapsulates its design and implements and offers interfaces to the outside, by which it may be composed with other components to form a large whole"* [3]. Other possible definitions emerged in the literature, e.g. [5].

To support component reuse in software development; several Component-based Development (CBD) methodologies have been proposed. A typical CBD methodology involves two main activities: developing reusable components and reusing components to develop software [8]. These two activities are also known as developing *for* reuse and developing *with* reuse, respectively. The latter attempts to solve the following problem: given a set of available components, which should be selected and how they be assembled to develop a component-based software system that satisfy a given set of requirements. Software development with reuse is receiving an increasing interest in the software research community (See for example: [4], [6], [7]). Developing for reuse focuses on developing the components that can be effectively used when developing a new software. This paper focuses on the problem of identifying reusable components that can be used over long period of time; a core problem in the activity of developing for reuse.

Business domain components are developed during the analysis of the business. A business domain component can be defined as: a software implementation of an autonomous business concept or process [2]. By encapsulating commonly used business concepts and processes, a business component enhance analysis and design within and across organizations.

Developing domain components can be challenging, however. This is because of the fact that business requirements may vary in nature even for the same application. Moreover, rapid changes in customer needs and business requirements may render business components obsolete if these components can not accommodate emerging requirements and changes with minimal cost. A Component that requires major changes to adapt to business evolution is said to be unstable components.

Several CBD methodologies have been developed over the last few years such as the Rational Unified Process (RUP) [9], the Catalysis [3], Select Perspective [10], and

978-0-7695-3596-8/09 $25.00 © 2009 IEEE
DOI 10.1109/ITNG.2009.276

the MaRMI-III [11]. Most of these methods identify components by first conducting requirements and domain analysis and developing a business model (also called conceptual model, or essential model [3]). This model forms a base for identifying and encapsulating business concepts in coherent components. However, none of the existing methods provides an explicit approach to develop stable component.

In this paper, we report our ongoing research in developing a framework that captures stable business components from business models. The proposed framework makes use of the theory of Formal Concept Analysis (FCA) [13] to identify business concepts and candidate components. In addition, the concepts of Software Stability Model (SSM) [14] are used to accomplish the stability of business components.

FCA was used to develop a framework to identify components from use-cases and class diagram [12]. The framework developed in [12] provides an approach to partition the class diagram into components, and hence, each class belongs to a component. Our framework, on the other hand, identifies possible candidate stable components that form a functional unit that is likely to appear in future applications. Moreover, our framework emphasizes stability as an important quality in the identified component.

The remainder of this paper is organized as follows: Section 2 gives a brief background of SSM and FCA. Section 3 describes the proposed framework. The framework is demonstrated in Section 4. Conclusion and future work are given in Section 5.

2. Background

This section provides a brief background on software stability and formal concept analysis (FCA) used in the proposed approach.

2.1. Formal Concept Analysis

FCA is a mathematical framework that can be used to represent and analyze data and their relationships [13]. It consists of two main components: *formal context* and *formal concept*. A formal context is a triple (G, M, I) where G is a set of objects [1], M is a set of features, and I is a binary relation from G to M. Given a set of objects $G = \{g_1, ..., g_i\}$ and a set of features $M = \{m_1, ..., m_j\}$; a formal context is represented as a table where each object forms a row and each feature forms a column. In this table, the intersection of the i^{th} row and j^{th} column contains an "X", if object g_i posses the feature m_j.

[1]This is different from the notion of objects in OO. In the subsequent Sections, we use "classes" instead of objects.

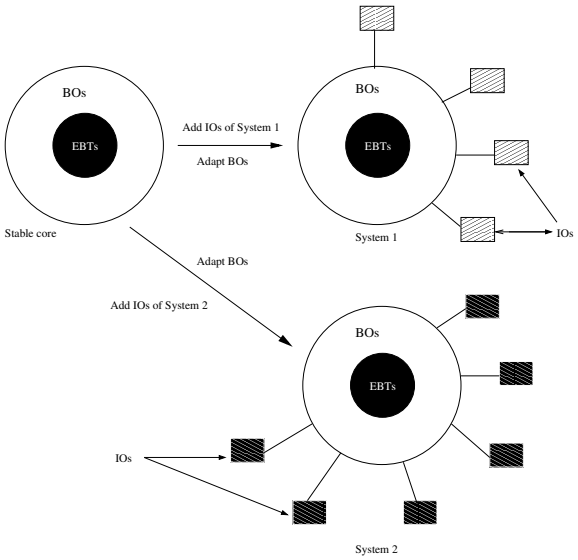

Figure 1. Software Stability Model

Let O be a subset of objects, i.e. $O \subseteq G$, then $\beta(O)$ is an operator that is defined as the set of features shared by all the objects in O.

Similarly, for a set of features F, where $F \subseteq M$, $\alpha(F)$ is defined as the set of objects that share all the features in F. For $O \subseteq G$ and $F \subseteq M$, a formal concept in the context (G, M, I) is defined as a pair (O, F), such that $\beta(O) = F$ and $\alpha(F) = O$.

Information represented with a formal context can be visualized by constructing the *concept lattice* of the concepts in the formal concept. Concept lattice can help in identifying useful notions between concepts such as the notion of *subconcept*.

If $C_1 = (O_1, F_1)$ and $C_2 = (O_2, F_2)$ are two concepts defined in the context (G, M, I), then C_1 is said to be a subconcept of C_2 if and only if $O_1 \subseteq O_2$(or equivalently, $F_2 \subseteq F_1$).

2.2 Software Stability Model (SSM)

Software stability Model (SSM) is a generic layered approach for modeling software [14]. SSM classifies classes that constitute the software into three strata (see Figure 1): *Enduring Business Themes* (EBTs); *Business Objects* (BOs); and *Industrial Objects* (IOs).

EBTs are classes that represent the enduring and core knowledge of the underlying business. Therefore, they are stable and form the nucleus of the SSM. In a banking system, for example, one possible EBT is "ownership"; without an "Ownership", there is no account.

BOs, on the other hand, are classes that map the EBTs

of the system into more concrete objects. BOs are semi-conceptual and externally stable, but they are internally adaptable through hooks . For example, an "Account" in the banking system is a BO.

Finally, IOs are classes that map the BOs into concrete objects. In our example, "SavingAccount" is a concrete representation of the BO "Account".

3 Proposed Framework

Figure 2 depicts the main activities of the proposed framework in the dot lines. The input to the framework is a set of requirements and use-cases. Since SSM concepts are used, it is expected that use-cases and requirements adapt the notions of EBTs, BOs, and IOs while conducting analysis. In the Step 1, use-cases and requirements are used to develop a business model and identify its scope. The business model consists of a collection of EBTs, BOs, and IOs, that are defined during requirements and domain analysis phases. The scope of the system is the boundary that isolates objects that uses the system from those who constitute the core system. An BO in the business model is said to be outside the scope of the system if it has no association with any EBT in the model (This concept is illustrated in Section 4).

In Step 2, the set of use-cases are partitioned into non-overlapping clusters based on their functionalities. The objective of use-case clustering is to group functionally related use-cases into one cluster that does a more general task. For example, a use-case to place an order and the use-case that describes the payment procedure for an order both can be grouped in one cluster named OrderItem.

In Step 3, the formal context of the problem is constructed by considering EBTs, BOs, and IOs as the object set, and the use-cases as the features set. Next, the concepts in the constructed formal context are computed. Concepts that are computed in this Step might be very large, and hence, to reduce the number of concepts, Step 4 is conducted. In this Step, a set of rules are used to filter the concepts and remove irrelevant concepts (defined below). The rules that are used for filtering can be adapted based on the application; however, we define a set of core rules that should be applied besides any new identified rules. These core rules ensure the compliance of the selected concepts with the SSM basic concepts. For example, in SSM, an EBT can not have a direct association with an IO; thus, a concept that contains only EBTs and IOs is deemed irrelevant concept. The five core rules are:

1. **R 1**: Remove unstable component;
2. **R 2**: Remove concepts that has no core EBTs;
3. **R 3**: Remove concepts that have objects that are outside the system scope;
4. **R 4**: Remove concepts that contain unconnected EBTs-BOs;
5. **R 5**: Remove subconcepts.

R1 excludes concepts that are not stable. We define a stable concept (or component) as a concept that contains n EBTs, m BOs, and k IOs, where $n \geq 1, m \geq 1$, and $k \geq 0$. Even though a collection of BOs and IOs may constitute a meaningful concept; however, such component in not considered stable and hence we do not include it in our candidate components. R2 identifies the core concepts of the domain. We differentiate between an EBTs and a core EBTs. An EBT is said to be a core EBT if it forms a standalone concept.

R3 excludes concepts that contain objects that are outside the scope of the system. Rule 4 deals with concepts that contain EBTs and BOs, but in the business model there is no real association between these BOs and EBTs. Such concepts do not reflect real business situations. Finally, R5 eliminates subconcepts (See Section 2.1).

The remaining concepts are then examined, during the Concept Assessment step, to determine whether or not they can qualify to be a candidate component. This conduct this step, we developed three metrics: Coverage Percentage (CP), Coupling Index (CI), and Stability Index (SI). A full definition can be obtained from [15].

The Coverage Percentage is computed for each component w.r.t. each cluster. It measures the percentage of the number of use-casse of a cluster j that is covered by concept C_i. To compute the CP of concept C_i with respect to Cluster j; the number of the overlapping use-case in Cluster j and C_i is divided by the number of use-case in Cluster j. The Coupling Index measures the average interaction between a given concept and the Clusters. $CI(Ci, j)$ can be computed by the following equation:

$$CI(c_i, j) = \frac{\sum_{\substack{\forall k \in Cluster \\ k \neq j}} CP(c_i, k)}{|Cluster| - 1} \quad (1)$$

Where: $|Cluster|$ represents the total number of clusters in the system.

The Stability Index measures the level of stability of the concept. The following is a generic formula for computing SI:

$$SI = 1 - \frac{\alpha|BO| + \beta|IO|}{|EBT| + |BO| + |IO|} \quad (2)$$

Where: $\alpha > 0, \beta \geq 1, |x|$ denotes the number of object in the layer x of the SSM. From our experience in using SSM concepts, typical values are: $\alpha = 0.5, \beta = 1$.

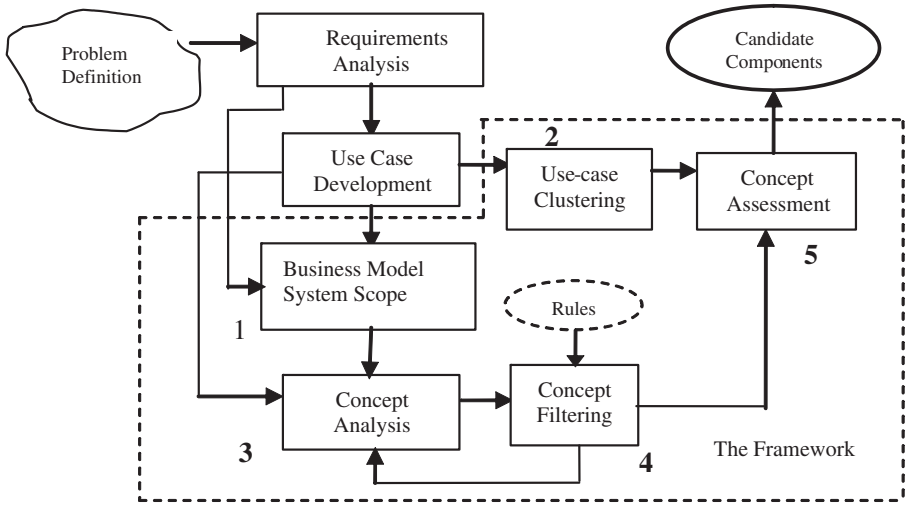

Figure 2. Main activities in the proposed framework for component identification.

Table 1. Portion of the formal context of the ESMS

Classes	Use-cases																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	X	X	X	X		X	X	X	X	X	X	X	X	X		X	X	X	X	X
B					X	X	X	X					X		X					
C	X	X	X	X														X		
D						X														X
E	X	X	X	X				X	X	X	X	X	X	X						

Clusters	Candidate Components	*CP*	*CI*	*SI*	*Q*
1	C1	33.33	12	1/4	92.16
	C2	66.67	12	1/3	171.34
	C3	100	12	2/5	248.00
	C5	100	18	3/8	238.25
	C8	66.66	18	2/5	175.32
	C9	100	21	1/2	254.00
	C11	33.33	10	1/3	106.66
3	C7	100	0	3/8	256.25
6	C6	33.33	7	3/8	115..91
	C8	33.33	23	2/5	103.66
	C9	33.33	32	1/2	109.66

Figure 3. Assessment of candidate components for Clusters 1,2, and 6.

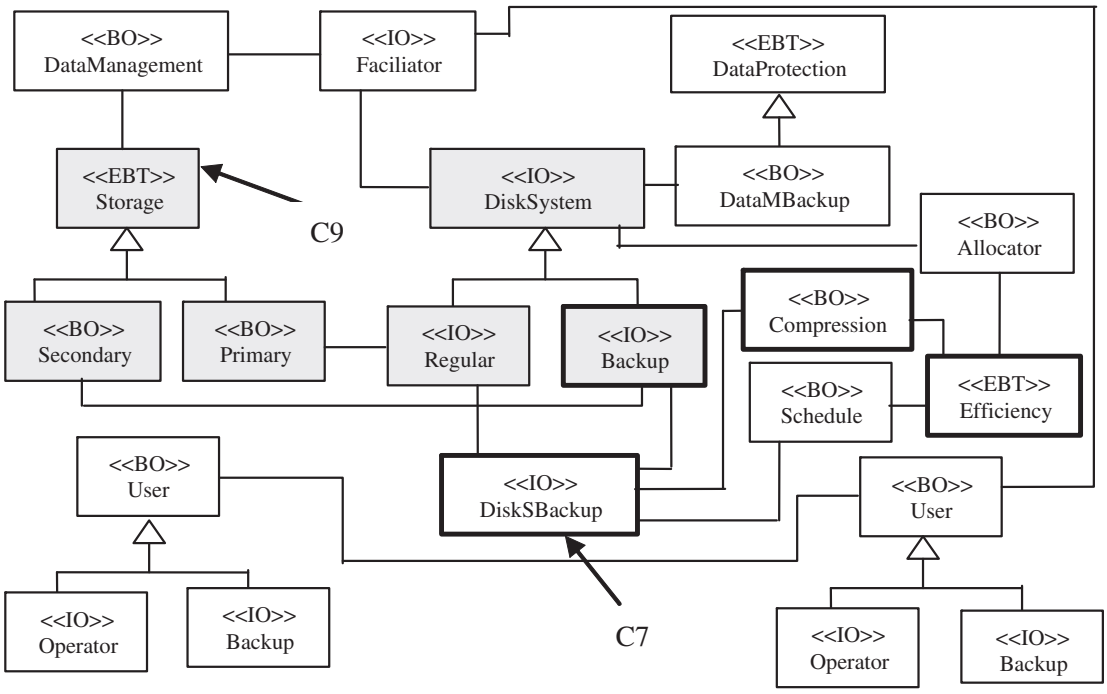

Figure 4. Main activities in the proposed framework for component identification.

After computing SI, CI, and SI, the overall assessment of the component quality,Q , is computed by the following equation:

$$Q = \mu(CP) + \sigma(SI) - \tau(CI) \quad (3)$$

Where: $\mu, \sigma, \tau > 0$. These factors can be customized to change the priorities of the qualities of the resultant components.

4 Case Study: The Enterprise Storage Management System

We illustrate the proposed through a case study of developing an Enterprise Storage Management System (ESMS). A complete use-case analysis as well as the design of the system can be found in [16]. The case study consists of 20 detailed use-cases and a business model with 23 classes (Figure 4). Portion of the formal context is shown in Table 1. Concepts were computed using the Concepts tool (available online at http://www.gaertner.de/ lindig/software/).

Objects are annotated by letters from A to W in the following order: EBT = Storage, Efficiency, DataProtection; BO= DataManagment, Allocator, Primary, Secondary, Compression, Schedule, DataBackup, User; IO= Facilitator, DiskSystem, Regular, Backup, DiskBackup, Disk, File, Computer, Client, Server, Administrator, Operator. Use Cases are numbered from 1 to 20, in the following order: Backup Data, Distributed/Centralized Backup, Manual Backup, Automatic Backup, Schedule Tasks, Archive Data, Perform Compression, Identify Resource, Switch Backup Disk, Delete Resource, Add Resource, Format Disk, Identify Bad Disk, Monitor File System, Minimize Space Waste, Create Reports, Report Backup Failures, Recover Data, Preserve Data, and Retrieve Data.

The 20 use-cases are grouped into 7 clusters:

(1) Backup Mechanisms={2,3,4}

(2) Data Operations = {1,6,18,19,20}

(3) Space Optimization = {7,15}

(4) Disk Management = {9,12,13}

(5) Resource Management = {8, 10,11}

(6) Report Management = {14,16,17}

(7) Scheduling = {5}

A total of 93 concepts were identified. Next, we apply the core rules to reduce the number of concepts. R1 is applied by removing irrelevant concepts ad discussed in the pervious Section. To apply R2, the core concepts are identified. In our example, based on the computed concepts, the core concepts are Storage and Efficiency.

To apply R3, we have to identify objects that are outside the scope of the system. In our example, the BO: User is outside the scope of the system as it is not connected to any of the EBTs. Next, R4 is applied removing concepts that have unrelated EBT-BO. Next, we apply R5 to remove sub-concepts. This can be easily done by examining the concept lattice of renaming concepts. Finally, we added a new rule (R6) to remove concepts that contain more than single EBT.

It should be noted that the more EBTs in the concept, the wider the scope of the concept. Wide scope may reduce the reusability of component. In addition, our case study contains only 3 EBTs, and hence, large components may include the whole system. Applying these six rules reduced the total number of concepts from 93 to 13.

To conduct the Concept Assessment step, we first computed the CP for every component with respect to every cluster. A concept with a nonzero CP with respect to a cluster is considered as a candidate component for this cluster. For each cluster, we compute the CI, SI and Q as shown in Table 3. In our study, we assume $\mu = 2, \sigma = 1.5, \tau = 1$ to compute Q , i.e., we prioritize the CP over the other factors.

If a concept forms a candidate for two different clusters, we chose the one with the highest Q. At the end, choosing which of the candidate component(s) is useful depends on different factors such as the nature of the project and the nature of the organization. We chose only two components out of the five possible components. These two components are represented by the two concepts C7 and C9. These two components are highlighted in the class diagram shown in Figure 4.

5. Conclusions and Future Work

In this paper, we presented a novel framework for identifying reusable components in business models based on the theory of FCA and the concepts of SMM. The proposed framework identifies candidate reusable stable business domain components that can be used to facilitate the development of future software systems. We demonstrated the proposed framework is through a detailed case study.

Currently, we are investigating the impact of different clustering schemes and filtering rules on the quality of the resultant components. In addition, more experiments are being conducted to better understand the impact of the different framework activities on the identification process and on the quality of the resultant components. Future extensions to this work include the development of a component description language (CDL) that can be used to formally specify identified stable components in order to facilitate their assembly in implementing new software systems.

References

[1] J.M. Buxton, P. Naur, and B. Randell, (editors) "Software engineering concepts and techniques" 1969 NATO Conf. of Software Eng.. NATO Science Committee, Brussels, 1969.

[2] P., Herzum, Sims, O. Business Component Factory - Johm Willey Sons, 1999.

[3] D.F. D'Souza and A.C. Willis, "Object, components, and frameworks with UML: the Catalysis approach. Addison-Wesley, 1998.

[4] P. Manolios, D. Vroon, and G. Subramanian, "Automating component-based system assembly," Proc. of Int. Symposium on Software Testing and Analysis, pp. 61 - 72, 2007

[5] C. Szyperski, Component software-beyond object-oriented programming. Readings, MA: Addison-Wesley, 1998.

[6] C. G. Haines, D. Carney, J. Foreman, "Component-Based Software Development / COTS Integration", SEI: Copyright 2007 by Carnegie Mellon University.

[7] I. Crnkovic, H. Schmidt, J. Stafford, and K. Wallnau. Automated component-based software engineering. Journal of Systems and Software, 74(1), January 2005.

[8] W-J. Lee, O-C. Kwon, M-J. Kim, and G-S. Shin, "A method and tool support for identifying domain components using object usage information," ETRI J., vol. 25, pp. 121-132, 2003.

[9] P.B. Kruchten, "The Rational Unified process: an introduction", Addison-Wesley, 2000.

[10] Select Perspective, available online at: http://www.selectbs.com.

[11] D-H. Ham, J-S. Kim, J-H. Cho, and S-J. Ha, "MaRMI-III: A methodology for component-based development" ETRI J. vol. 26, no. 2, pp. 167-180, 2004.

[12] H.H. Kim and D.H. Bae, "Component Identification via Concept Analysis," Journal of Object Oriented Programming, 2001.

[13] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival, editor, Ordered sets, Reidel, Dordecht-Boston, 1982, pp. 445-470.

[14] Fayad, M.E., Altman, A.: Introduction to Software Stability. Communications of the ACM, Vol. 44, No. 9, September (2001)

[15] H.S. Hamza "Towards an approach for identifying stable components in business models." Computer Science and Eng. Dept. University of Nebraska-Lincoln, TR-05-01-02, Jan 2005.

[16] Dan Glasser, Madeline Hardojo, Anand Sundaram, Nate Wells, Mohamed Fayad Enterprise Storage Management System available online at: http://designfest.acm.org/Years/2002.htm