Pattern-Based Coarse-Grained Component Modelling for Enterprise Application Software Developing

Hai Bo LI [a], Meng Xia LIANG and Ya Feng GAO

College of Computer Science&Technology, Huaqiao University,Xiamen,361021,China

Abstract. To improve the efficiency of ESA (Enterprise Application Software) development, component-based software development is a good solution. However, though those general software components, like JavaBean, COM/COM+ (Component Object Model), are maturely used to develop ESAs, it is still too inefficient to industrialize ESA development. This is because design of software component has a tightly correlation with the inner structure of an ESA and should not be independent from architecture of ESA. By analysing software patterns and characteristics, a business component (BC) modelling based on pattern is proposed first. Then a BC scheduling pattern and an executable workflow model are presented to show the principle by which BCs are organized and scheduled. The proposed method is a PSM (Platform-Specific Model) modelling which a part of MDA (Model-Driven Architecture) based framework. Finally, a case is showed that our method can provide theoretical and practical significance to the development software of ESA.

1 Introduction

Patterns of ESA (Enterprise Application Software) development have been evolved for several phases, from coupling code to data structure decades ago, and then separating code from database. Until now, object-oriented method is introduced to divide codes as many objects or components. At present, though SOA (Service-Oriented Architecture) presents more coarse-grained reuse mode than software component [1], it focuses on distributed applications [2]. To improve efficiency of building ESAs, the industrialized developing process still needs a method to construct coarse-grained component, even to be integrated into workflows.

The current research focuses on approaches based on MDD (Model-Driven Development). However, the MDD emphasizes the whole architecture of ESA functionality and behaviour, not the technology in which it will be implemented [3-5]. UML model can be considered as traditional software modelling for ESA developing [6], however, it lies in PIM level of MDA (Model-Driven Architecture) and can not be mapped into code directly. An object-oriented approach is needed for its advantageous modelling executable ESA, so as to attain a uniform overall approach to analysis, design, and implementation [7]. Though other object-oriented methods are provided for designing [8-13], executable component modelling is required for ESA developing nowadays.

In this paper, we propose a method of pattern-based coarse-grained component modelling for ESA developing (PCGCM). We first analyse the software patterns and characteristics of ESA, and then a BC (Business Component) scheduling pattern and an executable workflow model are presented in PCGCM. Next the principle is presented by which BCs are organized and scheduled by the execution workflow engine. The proposed method lies in PSM level in our MDA based framework-ICEMDA (Interoperable Configurable Enterprise Model Driven Architecture). Finally, a case is discussed using PCGCM.

The rest of the paper is organized as follows. In Section 2, the motivation of the paper is presented. In Sections 3, patterns and characteristics of component is analysed, and a case is discussed in Section 4. Finally, in Section 5, we present a summary.

2 Ideas of PCGCM

Relative to small-granularity components as elementary units in windows, such as those control objects, button, list and text box, etc., a big or huge-granularity component encapsulates many small-granularity components and process based on them to an integration entity. The big or huge-granularity component is considered as application (or application-like), and the process in it as micro process. This development pattern divides a large-scale software system into big or huge-granularity component and their integration. The big or huge-granularity components are called BC (Business Component), while the process between them is called macro process. From the point of view, ESA is a software system which composed of those applications and macro processes, and whose applications have same or similar

[a] Corresponding author: lihaibo@hqu.edu.cn

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

pattern. Namely, ESA represents the idea of application based on pattern.

3 Patterns and characteristics of BC

3.1 BC pattern analysis

ESAs show different patterns which are supported by different technologies and platforms. A comparison of different patterns is presented, as shown in Figure 1. Some pattern needs only one table, while another needs two or more table altogether. They all contain areas of query items and data presentations.

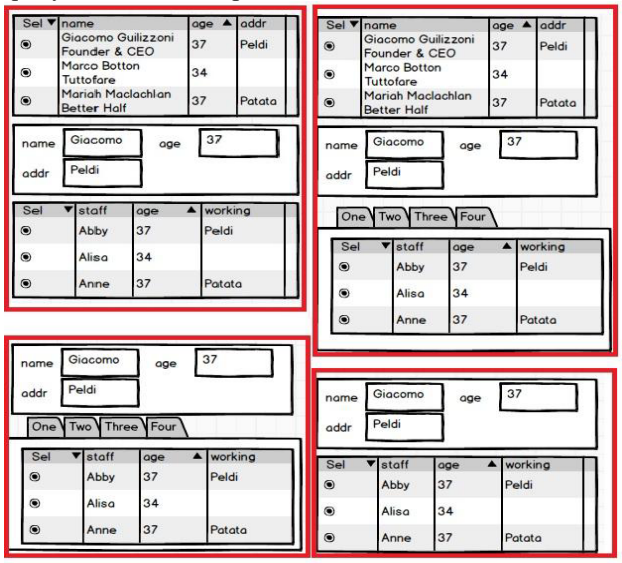

Figure 1. Several Traditional ESA Software Styles

In most ESAs, pattern is a kind of abstract from those common features which can be reused as template when developing. Pattern refers to application-oriented software pattern, whose characteristic is further divided into user-oriented (called user pattern) and system-oriented (implement pattern).

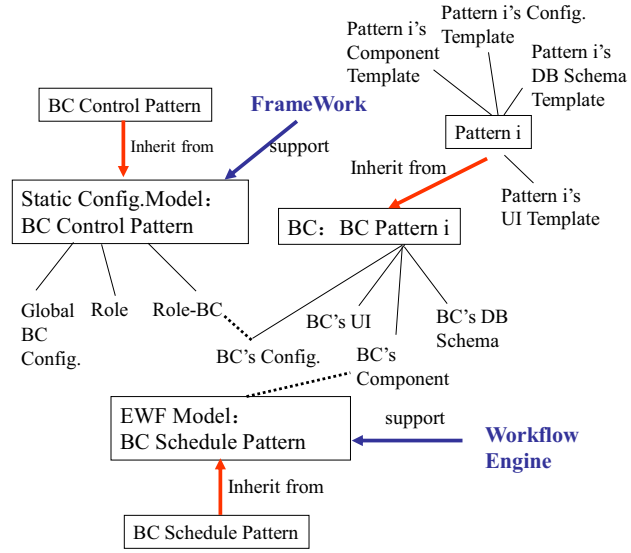

Figure 2. ESA Software Patterns

From view of user pattern, pattern represents interface style, operation style and control style. From view of implement pattern, pattern represents a common realization.

Relative to small-granularity component, BC is an independent application. BC pattern defines the style which inner structure of the applications should have. A BC application is composed of four parts, namely user interface, business logic, data pattern and configuration file, which describe BC's style and provide BC's template defining the common content of BC, as shown in figure 2.

As mentioned above, ESA is composed of more complete applications and macro processes, and participated by different roles. BC's control pattern defines how to control BC execution under system framework through access control. So the control pattern is also called framework pattern. BC's control pattern statically describes which BCs are used, which roles will participate and access associated with them. The framework determines which BC, and those small-granularity components such as button, data set and its attributes included in it, is executed by which role.

BC's scheduling pattern describes the execution sequence between each component, so to form macro process between BCs.

3.2 BC scheduling pattern

BC's scheduling pattern describes the execution sequences between each component, to form macro process between BCs.

In manufacture enterprise, for example, users can create an order from purchase requirement or purchase plan by clicking hyperlink to select them. The process involves three BCs, i.e. purchase order, purchase requirement and purchase plan. The process needs to be scheduled strictly.

First, users open the editing order interface, and then query, or skip the step. In any case, only those orders which are editing or have not summated to audit can be displayed.

Second, the purchase requirement and purchase plan interfaces opened by users only display those audited requirement and plan respectively. Those purchase requirement and plan which are edited or have not been audited can not be listed. Thus it can be seen that interaction between BCs is scheduled depending on state. Order audit is executed by a set of roles step by step. This audit process not only are constrained by audit rules, but also are controlled by states.

From the example, interaction between BCs have two aspects, data interaction and execution dependency (process). For the former, all the states of each BC are all determined. For example, when editing order, the selected purchase requirement must keep the state "audited". For the later, there exist several transfer patterns. For example, a state can only be transferred to its next state, depending on some conditions.

State transition produces event, and event triggers state transition depending on activity execution. For conveniently give expression to different state transition pattern, we adopt four workflow control pattern (WfMC)

[14]: sequential, and /or split and iterative. Under sequential routing, several activities are executed step by step. Under parallel routing, two or more activities are executed in parallel. Event triggers activity execution which must satisfy some constraint. This is ECA (Event Condition Action) pattern. An activity can produce many parallel events depending on some conditions or not, which is called event logic.

The items displayed on user interface are workflow task list (WfMC), which provides all tasks a user must do. For sequential state transition, an exact button can be determined. For example, button "audit" can be put on order edit interface. For other control patterns, next state(s) will depend on some conditions or not, so except button "next step", exact button can not be determined. An activity interface must own an exact state. State transition realizes BC interaction. This is workflow.

Some constraints may be satisfied before starting an activity when an event comes. For example, audit activity should satisfy such business rule: section chief audits orders if any value; if value>5000 and section chief has audited, then vice manager audits; if value>10000 and vice manager has audited, then manager audits. In the constraint, audit rule and audit state are described at the same time. Those business object attributes involved in the constraint and conditions used in event logic are called workflow relevant data. Because these constraint and conditions are used to control business logic, they are expressible in ESA. So workflow relevant data and constraint expression can be separated and converted to system configuration and modification.

3.3 Executable workflow model

Based on analysing above, executable workflow model should involve the four elements:

(1) State. State is used to control user interface, and to filter business object items as a part of query where expression.

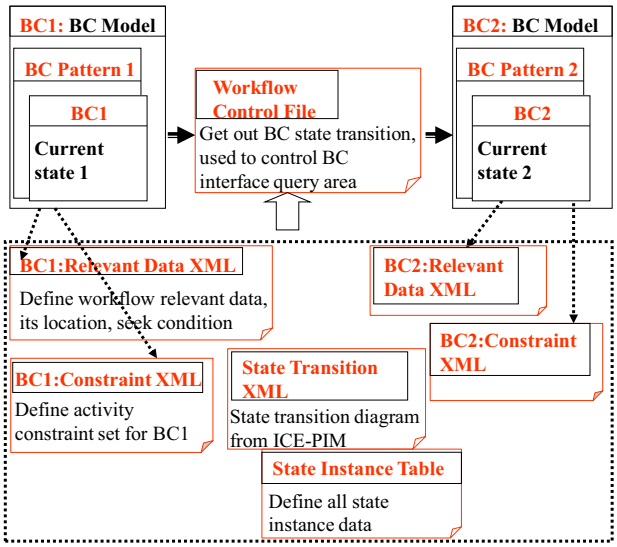

Figure 3. Executable Workflow Model

(2) Constraint condition. Constraint condition is constructed to executable SQL statements, associating with workflow relevant data in code. It is used to judge constraint of activity and condition to produce event.

(3) Workflow relevant data. Workflow relevant data refers to attributes which have association with constraint and condition expression used to control process.

(4) Workflow state instance table. State instance is used to track state transition. State instance is independent from framework and always stored in database.

Executable workflow model must configure these files at least, as shown Figure 3.

Workflow engine schedules process, as shown in Figure 4. When a user click a button in an interface, workflow engine computes the next state(s) according to workflow state instance and the current state, following the state transition diagram, then record the next state into workflow state instance table.

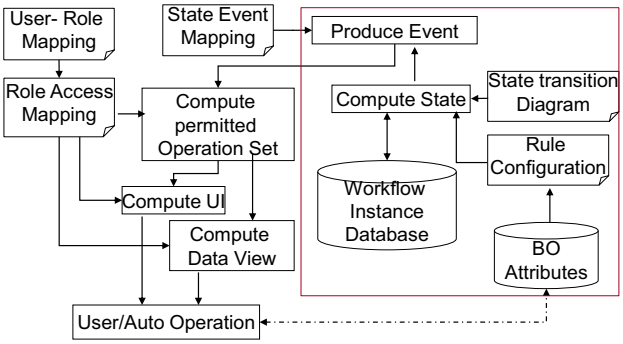

Figure 4. Workflow Engine Scheduling Process

The interaction between BCs involves two aspects: data interaction and execution dependency. Data interaction means that all user interfaces must have a determined state or state logic, which control business object items displayed. Execution dependency (process) means that when an interface is opened, and a business object instance is done, and next state needs to be computed. For example, an order is edited, and user clicks the button "audit" to request computing next state. Workflow engine receives the request, computes next state and updates the state instance table. After refreshing audit interface, the business object item will disappear.

PCGCM lies in the PSM level in our MDA-based framework ICEMDA, as shown in Figure 5.

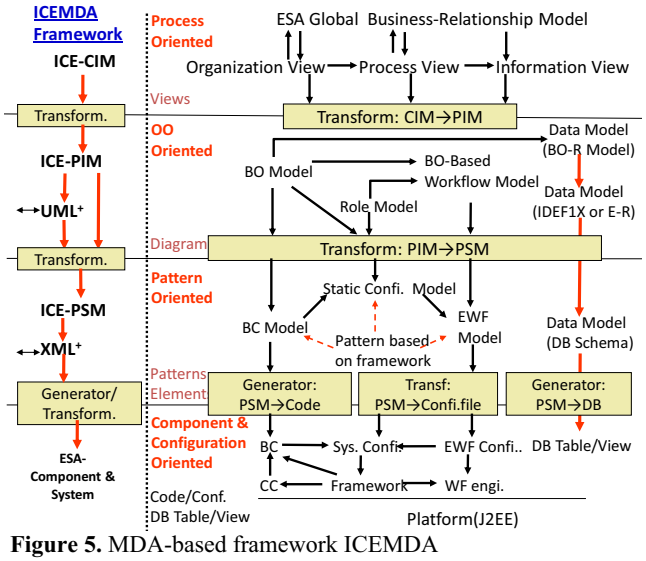

Figure 5. MDA-based framework ICEMDA

4 The case

The architecture of component based ESA can be seen in Figure 6. An execution of ESA can be divided into 3 stages. The first stage is constructing of component repository. In this stage, BC model which is descripted by PSM is transformed into execution component. Developers of components generate description files according to some standards, and store them into component repository all together.

The second stage is configuring of system. In this stage, some required components can be picked up from component repository, and then be initialized by some parameters.

The third stage is running of system. In this stage, besides J2EE application service, as basic executing platform, workflow engine and ESA framework are also needed. The workflow engine is used to access workflow configuration file, and the later is used to determine user roles. We have applied the modelling system and code generator toolkit to several ERP systems in Chinese enterprises. The efficiency of ESA developing is improved.

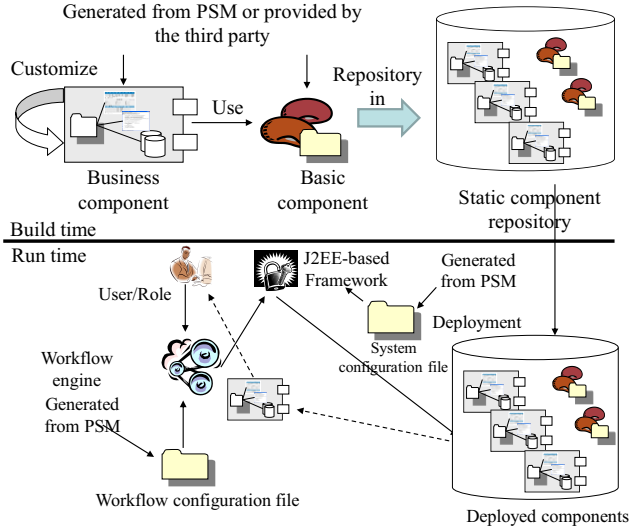

Figure 6. ESA Architecture Based on Component

Besides the patterns mentioned above, ESA developers can add their own patterns into to model, as shown in Figure 7.

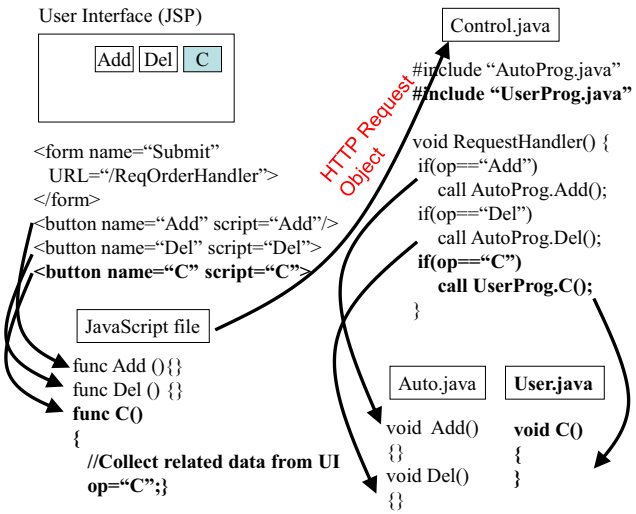

Figure 7. Users can add their code into the model

5 Conclusions

In this paper, a method of pattern-based coarse-grained component modelling for ESA developing, PCGCM is proposed to improve the efficiency of developing. The method presents a detail design for PSM in MDA-based framework. PCGCM can achieve this because it is object-oriented. Moreover, the principle by which BC is scheduled via an executable workflow model and its engine is represented. The proposed method has been applied to develop several ERP systems for Chinese manufacturing enterprises and has a good experience.

Acknowledgments

The authors are grateful to Prof. Dechen Zhan, Research Centre of Intelligent Computing for Enterprises & Service, School of Computer Science and Technology, Harbin Institute of Technology. This work was supported by Quanzhou Science and Technology Plan Projects (Nos. 2015Z125).

References

1. A. Khoshkbarforoushha, P. Jamshidi, M. F. Gholami, L. Wang, R. Ranjan. *IEEE Sys. J.*, **10**, 36, (2016)
2. H.B. Li, K.C.C. Chan, M.X. Liang and X.Y. Luo. *IEEE Trans.Indust. Inform.*, **12**,211(2016).
3. H. I-Ching, T. Der-Hong, H. Nien-Lin. *Comp. Stand. Interfaces*, **36**,648(2014)
4. Y. Rhazali, Y. Hadi and A. Mouloudi. *3rd Int. Conf. MODELSWARD*. 312(2015)
5. L. Zouhaier, Y. B. Hlaoui and L. J. B. Ayed. *IEEE 38th COMPSAC*. 535(2014)
6. Y. Rahmoune, A. Chaoui and E. Kerkouche.. *Procedia Comp. Sc.*, **56**,612(2015)
7. D. Ameller, X. Burgués, O. Collell, D. Costal, X. Franch and M. P. Papazoglou. *Inf. Softw. T.*, **62**, 42(2015)
8. G. Xie, F. Qian and X. Hei. *9th Int. Conf. CIS.* 688(2013)
9. D. Maplesden, E.Tempero, J. Hosking and J.C. Grundy. *IEEE Trans. Softw. Eng.*, 41, 691(2015)
10. S. Chandramohan, L. Ravikumar, G. Doolla, S. and S.A. Khaparde. *IEEE Trans. Power Syst.*, 30, 132(2015)
11. H. Fang, H. Zhu, J. Shi. *IEEE 16th Int. Conf. HASE.* 1(2015)
12. S. Al-Fedaghi. *IEEE 11th Int. Conf. ECTI-CON.* 1(2014)
13. A. Leva, D. Mastrandrea, M. Bonvini, A. V. Papadopoulos. *IEEE 7th Int. Conf. UCC.* 554(2014)
14. MC. *Workflow Management Coalition Specification: Terminology & Glossary. Document Number WFMC-TC-1011*. Brussels (1996)