J. Software Engineering & Applications, 2010, 3, 914-925
doi:10.4236/jsea.2010.310108 Published Online October 2010 (http://www.SciRP.org/journal/jsea)

BOSD: Business Object Based Flexible Software Development for Enterprises

Jindan Feng, Dechen Zhan, Lanshun Nie, Xiaofei Xu

Research Center of Intelligent Computing for Enterprises and Services, Harbin Institute of Technology, Harbin, China.
Email: Jindan.Feng@gmail.com, {dechen, nls, xiaofei}@hit.edu.cn

Received July 15th, 2010; revised August 3rd, 2010; accepted August 15th, 2010.

ABSTRACT

The enterprise software need adapt to new requirements from the continuous change management. The recent development methods have increased the flexibility of software. However, previous studies have ignored the stability of business object and the particular business relationships to support the software development. In this paper, a coarse-grained business object based software development, BOSD, is presented to resolve this problem. By analyzing the characteristics of variable requirement, business objects are abstracted as the separately-developed unit from business process, and are assembled to system through their relationships. The methodology of BOSD is combined with MDA (Model Driven Architecture) and implemented on the semiautomatic platform.

Keywords: *Business Requirement Change, Business Object, Relationship, Business Process, Information Systems, Flexibility, MDA*

1. Introduction

Enterprises use information system to optimize their management, and further enhance its competitive abilities in the markets. There are many changes in the optimization processes, such as the transformation from extensive management to intensive management, the transformation from various objects to uniform business object, and the transformation from disordered process to normative process [1]. One of the major challenges in the enterprise software is the adaptability at run time.

Currently, there are basically two types of factors to flexible software development. The first factor includes the potential changes of requirements and the disciplinarian of these changes. The researchers consider that the software adaptabilities focus on the business processes [2,3], and the processes follow the special change patterns [4]. The second factor is the proper development approaches which support the disciplinarian. There are three representative approaches to achieve the flexible software: Component-Based Software Development (CBSD), Model Driven Development (MDD), and Service-oriented Software Development (SOSD).

The CBSD emphasizes the software architecture, on which the software is iteratively assembled with different grained components [5]. The software developed by CBSD adapts to the changes of environment by reusing the existing components [6]. The MDD is an approach that generally separates the functionality from the implementation from the perspective of separation concern. The Computation-Independent Model (CIM), which is composed of business-centric models, is transformed to Platform-Independent Model (PIM) which represents software design rather than the details on the technology platform. The PIM is further refined to Platform-Specific Model (PSM), which is implementation model on the concrete platform. Finally, the software corresponding to the PSM is produced by code generator [7]. The MDD elevates the abstract level from codes to models. It makes the software quickly adapt to the new requirements through modifying the models. The approach depends on two aspects, the first, the well-formed models which have a directly effect on the correctness of the software [8]; the second, the mature technologies of the mapping and transformation among models [9]. However, the MDD cannot be separately used, and must be combined with the suitable modeling methodology to realize the specification of enterprise software [10]. The SOSD proposes that the service has a larger granularity than the source code traditionally, and the service is considered fundamental elements for developing applications [11]. The Business Process Execution Language (BPEL),which defines the sequence of Web Services Description Language (WSDL) interface, provides support for executable and abstract business process. Hence, the web service can be dynamic and flexible to meet changing business needs.

The above mentioned approaches are essentially classified into two categories: business process oriented software development approach and business object oriented software development approach [12].

The business process describes the ordering of activities for the purpose of achieving business objectives in the context of business organization and policy [13]. The business process oriented software development is an approach that focuses on the identifying business activities and decomposes the interaction of these business activities. A business process is performed by cooperation of a number of business resources called business objects. For an explicit business objective, these business activities are implemented and encapsulated into business objects according to the relationships of their functions [14]. The software based on business process can support dynamic configuration of activities with specific control styles such as the workflow [15].

The business object oriented software development is an approach that identifies business objects and the relationships between them. And then the business objects are assembled into the software as loosely coupled systems. A further benefit is that when the business process changes, the software can rapidly reconfigured through reusing the existing business objects to meet the changes at reduced cost of development and modification [16].

Although the approaches are considered business process-centric and business object-centric, respectively, they are complementary to the design of software, and always are used together [4,17,18]. In the combination of the processes and objects, one of them is the more changeable. As above mentioned, the business objects have more stability than the process. However, there are many different definitions of business object proposed presently. The general definition comes from Domain Task Forces (DTF) of Object Management Group (OMG). The Business Object is defined as a representation of a thing active in the business domain, including at least its business name and definition, attributes, behaviors, relationships, rules, policies, and constraints. A business object may represent, for example, a person, place, event, business process, or concept. Typical examples of business objects are: employee, product, invoice, and payment [19]. The business object can be used to describe the meta-model of enterprise [20]. A business object is also seen as a "class" or a set of classes in the object-oriented system, and further support a defined business area [21]. The existing definition of business object generally is considered any objects which are the inputs and outputs of business actions. The business objects generally refer to the classes which cover the contents in the different domains, such as business domain and software domain. In fact, the objects play different roles in the phases of business modeling, software design and software implementation etc. Therefore, business objects need to be distinguished explicitly.

In this paper, the concepts of business object and their relationships are proposed for the designing of enterprise software. We present (1) an explicit definition of business object; (2) the semantic completeness of business object; (3) business object is considered the smallest units of the business modeling, and the biggest units of the software realization; (4) a business component is an implementation of business object on the technology platform, and is a coarse-grained component.

In short, there lack of the strict and systemic methods to support the development of flexible software for enterprise. The potential requirements which occur in application phase need to be considered adequately in the software design phase. Aiming to adapt the more changes in the field of business management, the software adjusts itself locally by right of the reusability. The paper proposes an approach of software development based on business object, which is combined with the excellences of the mentioned approaches (CBSD and MDD). The approach is divided into three steps. First, we identify the business objects which have relative stability, and are regarded as separately-developed units of software. Second, we analyze the relationships among the business objects. Last, the business objects are composed into the flexible software under the control of the relationships mechanisms. The approach combined with the Model Driven Architecture (MDA) should be supported by a semiautomatic development platform. The paper attempts to provide the methodology for business modeling and its realization project for flexible software.

The structure of this paper is as follows. In Section 2, the variable requirements are briefly introduced. In Section 3, we present a definition of business object and the relationships between them. In Section 4, we further discuss the modeling methodology based on business object, and illustrate the details in our approach. The conclusion is given in Section 5.

2. The Variable Business Requirement

The changes of business requirements bring some negative effects on enterprise software at run time, sometimes even fatal impacts. For instance, 1) if a new data item is added and considered the primary key of the data view in the database, the software must adjust the relational components to adapt the change; 2) If the dependence relationships, (such as the state sequence of business documents, the associations between the documents), are changing in business system, the business process reengineering work through information technology is also needed for the new relationships. Therefore, if the software has abundant adaptability and flexibility, it can enhance the reengineering efficiency and decrease the reuse cost. In order to develop successfully flexible software for enterprise, it's primary problem for the software developers to understand accurately the business requirements and their changeable characteristics at run time.

In the real enterprises, business requirements always changes continually, and the changes follows the particular rules. According to our experiences in software development, we find that there are various changes of the business process between the different enterprises, which are in the same industry. In this section, we illustrate the variable requirements using the Purchase Order as an example. There are the different processes of making the Purchase Order, and every process represents a sequence of creating Purchase Order in **Figure 1**. The general routes is "submit Requirement → schedule Plan → inquire Price → build Order" in Process 1. The optional route shows that Purchase Order is refined directly from the Requirement Bill in Process 2, and the route is used for emergency requirements. With the improvement of information degree in enterprise system, the contents of Purchase Requirement in Process 4 can be imported from the records of the material requirements in the workshops, or obtained from computing the replenishment amount for inventory. The different types of Purchase Requirements, such as large facilities and production materials in Process 4, will go through the different approval processes.

Although there are different processes which are used for creating the Purchase Order, the business documents in the processes are limited to a certain scope, and they are Requirement, Plan, Quotation Invoice and Order for purchase. And then, the corresponding forms in each Process have similar kernels, for example, the every Requirement Bill contains the data items as follows: keys, Need Department, Item Code, Need Amount and Date etc. Sometimes, these data items may have the different names, but the same semantics. According to the standardization degree, the assistant information in documents may exist or not, such as the data items used for describing state transformation. Therefore, we find that the documents are similar, and there are little differences between them in the same business transactions.

From the analysis above we can know, the processes of making purchase order are various, and essentially the relationships between the documents are various. It's obvious that the business documents have relative stability. The business documents are more stable than the business processes. By analyzing the feature of business process, we find that the processes also follow the rules. The relationships between business objects are divided into three types: approval process, association process and integration process, and they are shown in **Figure 2**. The approval process is one of the basic processes in the lifecycle of business documents. The approval process is a set of activities, including auditing and affirming. The people at the different ranks run the activities corresponding to their rights. The association process is composed of the activities which deal with the many-to-many relationship between two documents. The integration process is an activity set to create new documents through the computation based on one or more existing documents. The computation rules are always complicated, and need to be produced by hands. If these relationships are identified clearly, they are implemented based on the particular patterns. Therefore, the difficulty of assembly is reduced to a certain degree.

The business system is thought as dynamic network,

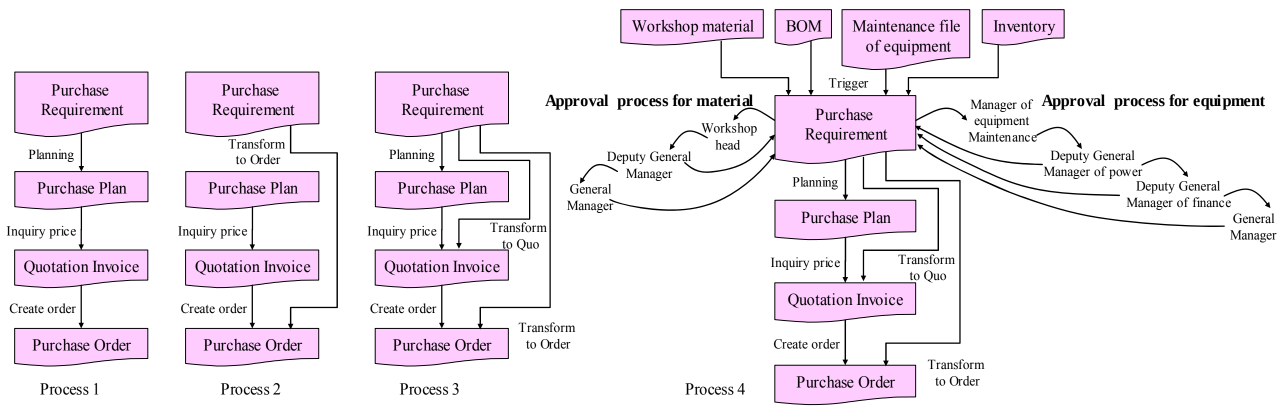

Figure 1. The optional processes of creating purchase order.

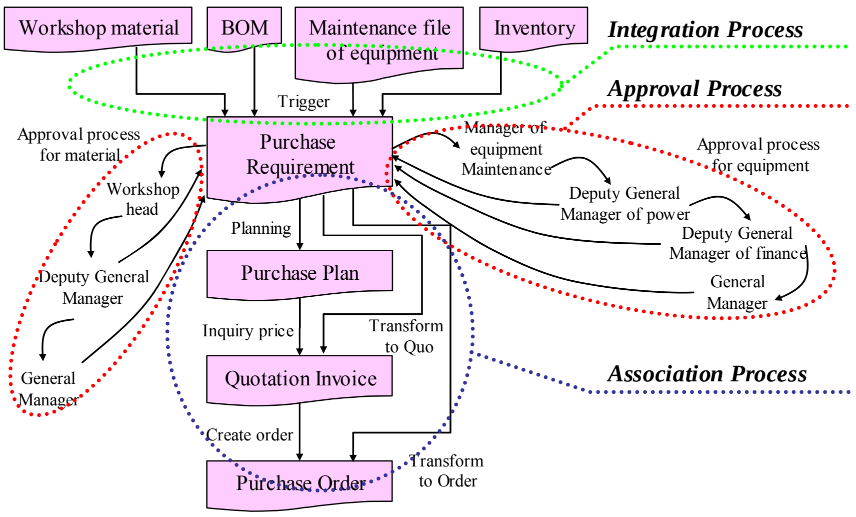

Figure 2. Three types of business processes.

and is composed of stable business documents and the various business processes. If the software has been implemented only using the approach based on the process, it is difficult to adapt quickly to the new relationships. Therefore, the approach is combined business objects with business processes together in this paper. The process-oriented requirement analysis is transformed into the object-oriented software development in the approach. First, requirement analysts understand the business processes in the enterprise, and identify the business documents. Second, the business objects are abstracted from the documents by software designers. The objects are as independent as possible, thus they can be reused for the variable processes. Third, the connections between objects are built in the light of current requirements. Finally, the business objects are realized to business components. The business processes are transformed to the cooperation relationships among the components, which are supported by the workflow engine. By doing so, we enable to assemble the flexible software form the business components and workflow engine.

3. Business Object

Basing on the business object from OMG, the business documents and their relationships are identified to business object and these three relationships in **Figure 3**, respectively. And then, we separate the business semantics of business object from its implementation.

Hence, our business objects differ from the objects of OMG in the following aspects: 1) from the perspective of software design, business object is a basic operation unit of system, is a integration of the data and the operations on them; 2) from the view of external application, business object is an integrated body, including the information which are collected and transformed by the users, the state transformations and operations which affect on the information, and the state transformation is such as start, running, end; 3) from the view of object-oriented structure, business object has an identifier number, data sets, operation sets, and its own lifecycle; 4) from the view of representation format, the business object represents that some children tables are attached to a father table; 5) from the software implementation view, business object is implemented to business component with the software technology.

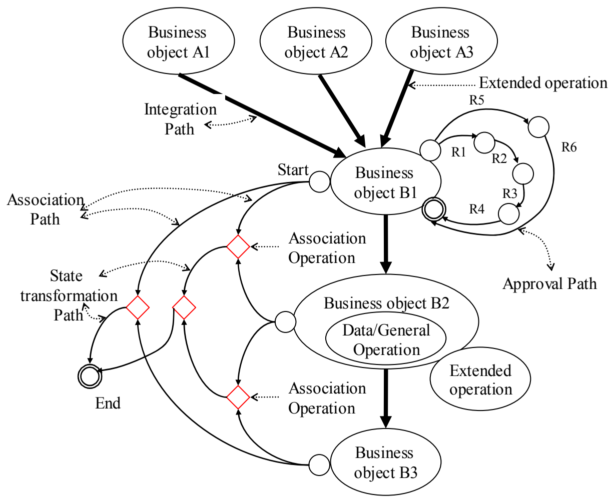

Figure 3. The business objects and their relationships.

The definition of business object is as follows:

Definition 1: ***Business Object*** **(*BO*)** is defined as a basic operation unit with the integral semantics in the field of enterprise business. Business object is described as *BO* = (*BOID*, *DS*, *OPS*, *SS*), where:

1) *BOID* is the *ID* number of *BO*;

2) *DS* is the data sets. $DS = \{CDS, \{ADS\}\}$; *CDS* is the core data set; *ADS* is assistant data sets. Every data set is composed of numerous data elements, which have the same or similar semantics. *DS* need satisfy the following constraints: ① $CDS \neq \emptyset$; ② d is a data element of the *DS*. $\forall BO$, $\exists X \subseteq CDS$, $X = \{d_1, d_2, d_3, \ldots\}$. But $\forall X$, there doesn't exist more than one x_i, $x_i \in X$, which uniquely determines the *CDS* of *BO*; ③ $ADS = \{ADS_1, ADS_2, \ldots, ADS_i, \ldots, ADS_n\}$, $ADS_i = \{d_n \mid d_n \textit{ is data element}, d_n \in DS\text{-}CDS\}$ and $ADS_i \cap ADS_j = \emptyset$, where $i, j = 1, 2, 3\ldots$; ④ $CDS \cap ADS = \emptyset$; ⑤ If $ADS = \emptyset$, there is $DS = CDS$;

3) *OPS* is the operation sets of *BO*. $OPS = \{GPS, EPS\}$, $GPS \cap EPS = \emptyset$, $GPS \neq \emptyset$. *GPS* is general operation set, *EPS* is extended operation set. An operation set is composed of many associated operations. $GPS = \{op_1, op_2, \ldots, op_i, \ldots, op_n\}$, where $i = 1, 2, 3\ldots$; *OPS* need satisfy the follow constraints: ① $GPS \neq \emptyset$; ② *op* is a operation element of the *OPS*. $\forall BO$, $\exists Y \subseteq GPS$, $Y = \{op_1, op_2, op_3, \ldots\}$. $\forall Y$, there exists single one $y_i \in Y$, *that* makes *GPS* exist uniquely; ③ If $EPS = \emptyset$, then $OPS = GPS$; ④ *GPS* acts on the *CDS* at least, else *EPS* deals with *CD* or *ADS*.

4) *SS* includes the states and their transformation. $SS = \{(S_i, T_i)\}$, where $S_i = \{s_{is}, \ldots, s_{ij}, \ldots, s_{ie}\}$, s_{ij} is the middle state in the S_i. $\forall S_i$, there exists a start state called s_s and a end state named as s_e; $T_i = \{ t_{ij} \mid t_{ij} = (s_{im}, s_{in}, p), op_k \in p, p$ is a ordered set of op_k, $p \subset OPS$. t_{ij} represents that *BO* is transformed from the present state s_{im} to the next state $s_{in}\}$.

BO is classified into different categories according to their business functions in the certain industry, for example, the class of *Purchase Requirement*, the class of *Equipment Maintenance*, the class of *Warehouse Entry* etc. Every category of BO is an abstract set of business documents, which have the similar features. One of the classes is further divided into children classes according to the concrete semantics. For instance, the class of *Purchase Requirement* has children classes as follows: the requirement for production material, the requirement for equipment, the requirement for office etc.

BO describes all of the semantics at the range of business document. The semantics is partitioned into some semantic dimensions. Every semantic dimension contains three planes: data, operations and states. Every plane is made up of two axes. The data plane is $P(x, z)$, operations plane is $P(x, y)$, and states plane is $P(y, z)$. Thus, *semantic dimension* is defined as *BSD* (*id*, *BOID.ds*, *BOID.op*, *BOID.ss*), where $BOID.ds \subset DS$, $BOID.op \subset OPS$, $BOID.ss \subset SS$. When the operation is triggered in one dimension, the values of data and states will be changed in the same semantic dimension. **Figure 4** gives an example to illuminate what is semantic dimension in BO. The main semantic dimension of *Purchase Requirement* is a cross section, which is composed of *CDS*, *GPS* and *Running state*, where *CDS* = {*Bill identifier*, *Material item*, *Requirement amount*, *Requirement date*, …}, *GPS* = {*Insert*, *Modify*, *Delete*, *Release*, *Perform*, …} and *Running state* is *set* (S_{00}, T_{00}) = {{*NewCreated state*, *Released state*, *Running state*, *Finished state*}, {(*NewCreated state*, *Released state*, *Release*), (*Released state*, *Running state*, *Perform*)…}}.

We proceed to define the three relationships based on the semantics partition, which are shown in **Figure 2**. The definition of approval relationship is as follows:

Definition 2: ***Approval Relationship*** is defined as a set of *AP*, is a series of the approval actions. The approval action is represented as a node. The *AP* = (*StepID*, *Precondition*, *Roles*, *Operation*, *Next StepID*), where *StepID* is the node identifier. This node is visited when the precondition has been prepared. *Roles* = {*role* | *role has the rights to perform the approval operation*}. *Operation* = { *a* | *a represents op* $\wedge$ (*a* = *accept or a* = *reject*)}, *Next StepID* = { *b* | *b represents StepID*, where *b* = *x iff Operation* = *accept*, *or b* = *y iff Operation* = *reject*, *x*, $y \in$ {*StepID*}}.

After the auditing people finish the operations, the values of auditing dimension will be updated, and than the BO will be transformed to the next steps, in where the transformation path are predefined in *AP*. There exist many different approval paths for a BO, because the enterprises apply the extensive or intensive management pattern.

The auditing relationship is an ordered route in the auditing dimension. For instance, the approval dimension involves the following elements: ADS_i = {*approval action ID*, *people name for approval*, *approval date*, *advice*}, EPS_i = {*Query*, *Accept*, *Reject*} in the approval $action_m$, where $m=1,2,3,\ldots,n$. (S_{ij}, T_{ij}) = {{*Waiting State*, *Accepted State*, *Rejected State*}, {(*Waiting State for Approval*, *Approving*, *Accept in the action*$_1$),…, (*Approving*, *Accepted State*, *Accept in the action*$_n$)}}.

The approval relationship has effects on the inner of one BO. On the contrary, the association and integration relationships are built on the corresponding semantic dimensions among Business Objects (BOs). When two BOs are connected with the relationships of association or integration, the any operation of BO may change the state of the other BO.

For the association relationship between two BOs, a middle BO named as *Association Object* bridges many-to-many relationship of data. In **Figure 4**, BO3 is an association object between BO1 and BO2. The association object is a virtual object for software designer, is not an entity which is transacted by the business people.

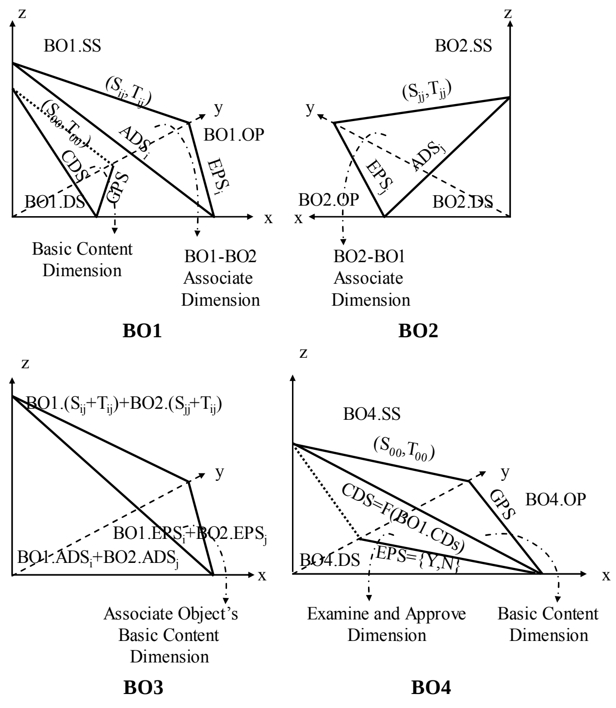

Figure 4. Business semantic dimensions.

Definition 3: ***Association Relationship***.

$\forall BO_1$, $BO_2 \in BO$, if BO_1 is associated to BO_2, the following condition must be prepared:

1) There is BO_2-oriented semantic dimension named as BSD_1 ($BO_1.ASS$, $BO_1.ADS_i$, $BO_1.EPS_i$, BO_1. (S_i,T_i)). At this moment, the every element of this tuple has special semantics. The data element of $BO_1.ADS_i$ is associated to the data element of BO_2. The EPS_i is an operation set, which connects the data mapping from BO_1 to BO_2. $BO_1.(S_i,T_i)$ represents the association states in this dimension of BO_1, whose values will be changed because of this mapping.

2) BO_2 also has the BO_1-oriented semantic dimension, named as BSD_2 ($BO_2.ASS$, $BO_2.ADS_j$, $BO_2.EPS_j$, $BO_2.(S_j,T_j)$), the others are similar to (1).

3) There exists one association object, represented as BO_3 (BO_3, $BO_3.DS$, $BO_3.OPS$, $BO_3.SS$), Where $BO_3.CDS = BO_1.ADS_i \cup BO_2.ADS_j$; $BO_3.GPS = BO_1.EPS_i \cup BO_2.EPS_j$; $BO_3.SS = \{(S_0, T_0), BO_1.(S_i, T_i), BO_2.(S_j, T_j)\}$, it is composed of the *Running States* (S_0,T_0) in BO_3 and association states.

The core data set of association object is composed of data in these BOs which associate with each other. There are three types of associations according to the functions of data items, such as the association between data items, the association between keys, the association between keys and attributions. In association object, the *ADS* contains data elements as follows: *Creator*, *Creating date*, *the Last Modifier*, *the Last Modifying Date* etc. In the complex system, the association is bidirectional and transmissible. For example, the Process 1 in **Figure 1** shows that there is a direct association between the *Purchase Requirement* and *Purchase Plan*. The numerous records of materiel requirement are combined into one purchase task of *Purchase Plan*. And then the *Purchase Order* is transformed from the *Purchase Plan*. Thus, the indirect association between the *Requirement* and *Order* is connected. The object of *Purchase Requirement* can build many direct or indirect association paths with the other objects in its lifecycle. The subsequent records in the business process can be traced to the sources along the paths. By doing so, the intensive management keeps at the fine granularity in the enterprise.

Definition 4: ***Integration Relationship.***

$\forall BO_1$, $BO_2 \in BO$, there exists $BO_1(BO_1$, $BO_1.DS$, $BO_1.OPS$, $BO_1.SS)$ and $BO_2(BO_2$, $BO_2.DS$, $BO_2.OPS$, $BO_2.SS)$. There is not less than one $d = F(BO_1.d_1,\ldots, BO_1.d_n)$ in $BO_2.CDS$, where F is formula used for computing value of BO_2. Thus, there exists the integration from BO_1 to BO_2.

In real enterprise, the part values of business object are obtained from many business objects through the complex computation. Therefore, the definition 4 can be extended. If there are integration relationships in the BO_1, BO_2, …, BO_n and BO_j, BO_j (BO_j, $BO_j.DS$, $BO_j.OPS$, $BO_j.SS$) must satisfy that there exists no less than one data item $d = F(BO_1.d_1, \ldots, BO_i.d_k, \ldots, BO_n.d_n)$, where F is formula used for the computation from BO_i to BO_j, $1<i<n<j$. Generally, the formula F is various in different enterprises. Therefore, the implementation of F is second development by the programmers. The operation patterns of integration are classified into *Push Integration* and *Pull Integration*. The pattern called *Push Integration* need satisfy the following constraints: 1) the lifecycle of BO_1 is earlier than BO_2; 2) the operator in BO_1 is used for triggering the integration; 3) the BO_1 starts the integration, and BO_2 receives the messages. On the contrary, the *Pull Integration* is that the trigger operation belongs to BO_2, and BO_2 is defined as active object.

4. BOSD

4.1. Basic Thinking

In this section, we propose the software development approach that combines business object and business process together. The basic thinking is shown in **Figure 5**.

In the phase of requirement analysis, the consultators firstly analyze the management patterns and further build the models through researching the actuality of enterprise. First, the decision model is planar model which is an extended GRAI-GRID model. The decision model is

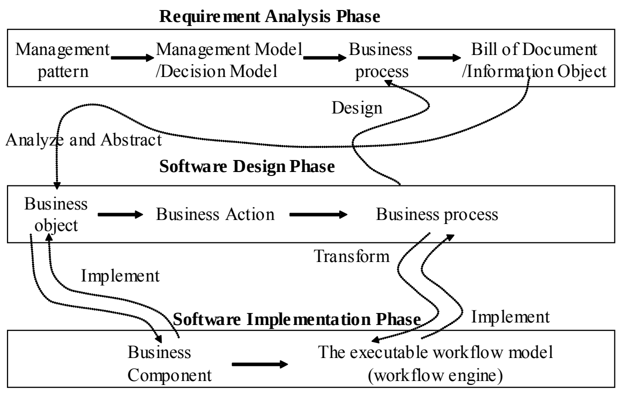

Figure 5. The process of BOSD.

defined as *Decision Model* = (*H/P*, *FUNC*, *ORDER*, *DecisionCenter*), where, *H/P* is row and *FUNC* is columns; *H/P* represents the different decision granularity, and is used for defining the time span (such as year or month) or different decision layers (such as stratagem layer, tactics layer and operations layer); A business target is generally realized by the collaborative activities, each of which is executed respectively by the different departments. *FUNC* is a set of functions. *FUNC* is described as a set of relationships, which are used for realizing a common target. The *DecisionCenter* is a cross cell of the row and column, and is defined as $\{order \mid order = f(h/p, func),\ h/p \in H/P,\ func \in FUNC\}$. The *order* represents an instruction, which is always saved in the business document. Second, we can produce the business process models through tracing the physical transformation from the materials to the products. The business process in the requirement analysis phase is defined as follows: *Process Model* = (*IP*, *ORG*, *DOC*, *ORDER*, *ACTION*'), where the IP is the information entity, represents an instruction or document in process model. The $IP = \{ip \mid ip = order \text{ or } ip = doc \text{ and } order \in ORDER,\ doc \in DOC\}$. The *ORG* represents the department set. *DOC* is a set of documents. The *ACTION'* defines the set of activity, which is executed by the *ORG* and acts on the *ORDER*. Finally, the information objects are abstracted from business documents which are used in the process models. Thus, the function-oriented requirements are collected clearly.

Aiming to enhance the independence of business object, the functions acting on the information objects are combined with the information objects to business object in the phase of software design. The business objects are connected to business processes by the business actions. For designing the flexible software using this approach, the approval, association and integration relationships are implemented by the particular approach in **Figure 6**. The software development is followed these steps: 1) the business component corresponding to the special BO is developed independently; 2) the inner relationships, such as the approval process and state transformation, are firstly considered to implementation; 3) when the whole business object is designed clearly, we connect them using the association relationship. And we define the state transformation which is affected by the associations under the control of association mechanism; 4) at last, the complicated integration relationship is developed by the extended mechanism, including automatic data import, computation and carrying forward etc.

In the development phase, business components are transformed from BOs, the relationships between them are implemented to executable workflow model. Thus, the workflow model is defined as (*BO.DS*, *BO.AT*, *CON*, *ATRelation*, *RoleSet*), where the *BO.DS* is the set of data; *BO.AT* represents the activity set, is mapping to the *BO.OPS*. $CON = \{condition_{x\to y} \mid condition \text{ is the transformation condition from the activity } x \text{ to } y\}$; The *ATRelation* is the transformation rules, is defined as $ATRelation = \{f \mid y=f(x) \text{ iff } condition_{x\to y}=true,\ x\in BOi.AT,\ y\in BOj.AT,\ i,j=1,\ldots,n\}$; *RoleSet* is the roles which take part in the *BO.AT*.

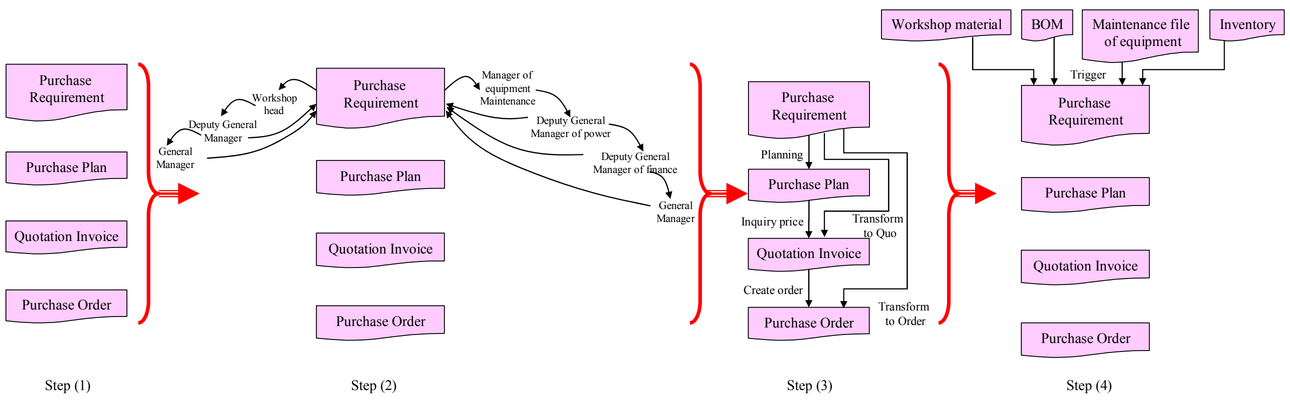

Figure 6. The steps of building the relationships between BOs.

4.2. Development Framework

The methodology introduced in Section 4.1 is very enormous and need to be supported by software technology. Hence, we choose the platform named as ICEMDA (Interoperable Configurable Executable Model Driven Architecture) [22,23] to implement this thinking. The platform has been developed mostly and is shown in **Figure 7**. There are five phases using this platform to develop the software for enterprise as follows: building the CIM for business requirement, building the PIM for the software design, and building the PSM for implementation on particular platform, generating the components from PSM, and assembling the components to a system.

The models in the requirement analysis phase are structured as decision model, business process model, information model and organization model on the support of using graphical modeling tools [24] on the CIM layer. The models on the CIM level are semi automatically transformed to the models on the PIM level, which involves business object models, BO-based workflow model, role-dependent model and data model. The three relationships above mentioned are described detailedly in the BO-based workflow model on the PIM layer. And then the PIM is refined to the PSM on J2EE platform, such as business component model (the model for component implementation) based on a specific software pattern, the executable workflow model, and the deployment model which defines the assembly of business component and workflow model. The business component is produced from the Platform-specific component model by code generator [25], and then can be secondly developed for meeting the special requirements. In the system for enterprise, the workflow engine parses the workflow models, and then choreographs the components to supply the different services to the clients.

4.3. BO and Relationships Expression

We illustrate the expression of BO and the relationships using a purchase management system. The business objects of purchase system are *purchase requirement*, *quotation*, *purchase plan*, *purchase order*, *arrival notice* and *advice of settlement* etc.

According to the definition 1, the contents of BO are described by the graphical platform-independent models in **Figure 8**. There are four types of platform-independent models which correspond to the data, state and operation set in Definition 1, and they are the data diagram, state diagram and class diagram. The use case diagram is mapping to the role-dependent model, which represents the roles act on the business objects in the workflow model.

The relationships between BOs in the Section 3 are described by a special relationship object called BOR (Business Object Relationship). The BOR can be implemented into three instances: association object, auditing object and integration object. And the BOR is also made up of the rules from the views of the data, operations and states. We can define the BOR = $\{f_R(BO_i, BO_j) \mid f_R = (r_1 \wedge \ldots \wedge r_k), r_1, \ldots, r_k \in R\}$, $R = \{ r \mid r = f(xs_m, xs_n), xs_m \in BO_i.XS, xs_n \in BO_j.XS, 1 \le m \le |BO_i.XS|, 1 \le n \le |BO_j.XS|$, where $x = d \mid op \mid s, X = D|OP|S \}$. The Process 4 in Figure 1 is modeling to the class diagram, which is shown in Figure 9. The bold lines represent the source process, and every BO has its own auditing object, just like the Auditing 2 for materiel. The graphical model is used for understanding easily, and further the model is

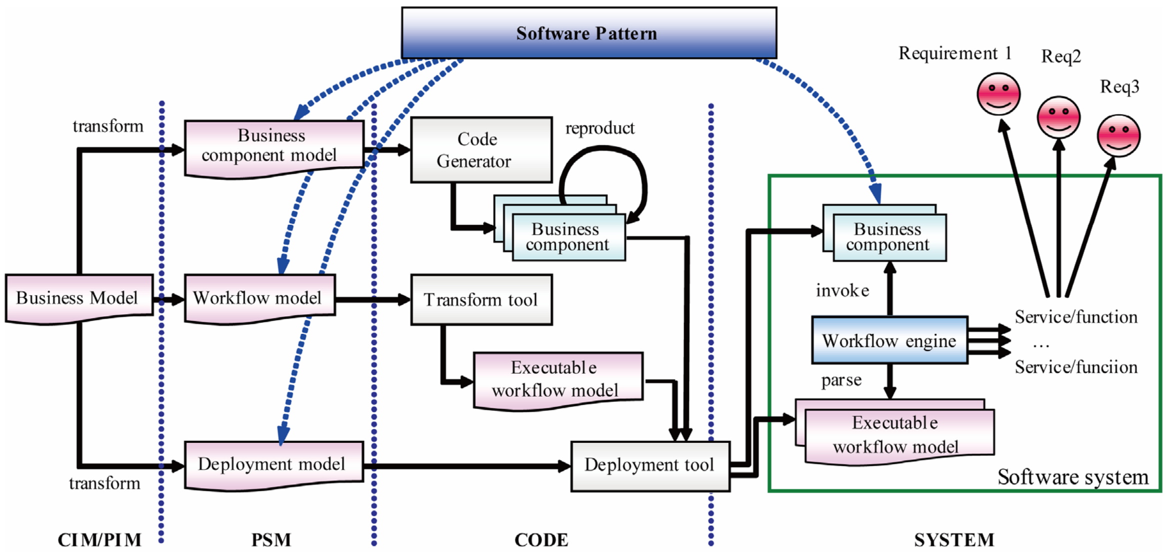

Figure 7. The ICEMDA framework.

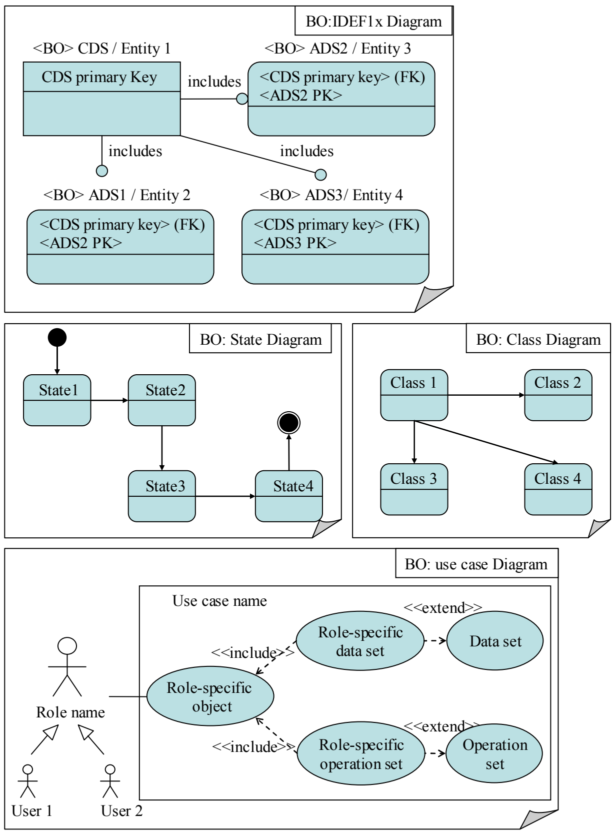

Figure 8. The BO platform-independent model.

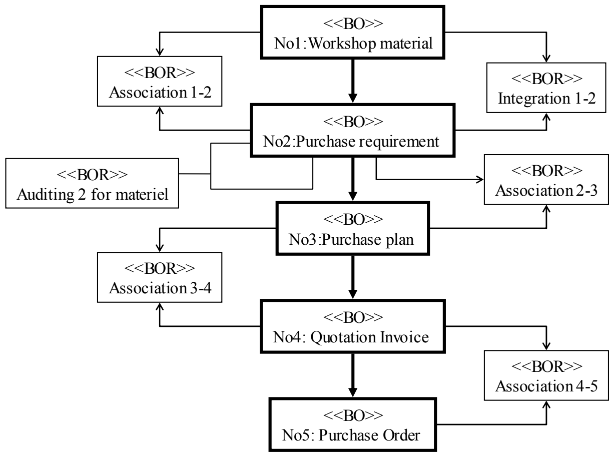
<<BO>> No1:Workshop material
<<BOR>> Association 1-2
<<BOR>> Integration 1-2
<<BO>> No2:Purchase requirement
<<BOR>> Auditing 2 for materiel
<<BOR>> Association 2-3
<<BO>> No3:Purchase plan
<<BOR>> Association 3-4
<<BO>> No4: Quotation Invoice
<<BOR>> Association 4-5
<<BO>> No5: Purchase Order

Figure 9. The class diagram of BOs.

saves as the file in XML (eXtensible Markup Language). Thus, the *BOR* is expressed as the fragment of XML.

When the *Purchase Requirement* is arranged into *Purchase Plan*, the association relationship between them has been built. It's defined as the following XML list:

```
<associationSet>
  <ass AssBOId="No2-No3">
  <AssedObject BOId="No2"
  name=”Requirement”
     assString="Need_code,Material_code,
     Material_name,needNum">
     </AssedObject>
  <AObject BOId=" No3"
  name=”Purchase Plan”
     assString="Plan_code,Material_ID,PlanNum">
     </AObject>
  <assCondition>
     Material_code = Material_ID
     </assCondition>
  </ass>
</associationSet>
```

The second example is that *Purchase Order* is approved by the appropriate roles, each of which has the permission right according to the material type or money amounts. Thus the approval relationship of *Order* can be described as:

```
<ApprovalPath BOId="No5">
  <step StepId="start"
     personString="dept manager"
   condition="money<=50000 and money>0 or material belongto 'steel'"
    nextStep="accept?first:goback"></step>
  <step StepId="first"
    personString="assistant manager"
    condition="money<=100000
    and money>0"
   nextStep="accept?second:goback">
   </step>
  <step StepId="second" ...></step>
  <step StepId="end" personString="manager"
   condition="money>100000"
   nextStep="accept?gonext:goback"></step>
</ApprovalPath>
```

The third example: when the material in workshop lacks abruptly, the record of *Requirement No2* is achieved through importing the data of *Workshop Material Requirement.* For supporting this process, that integration relationship between these objects need be predefined. When the delivery date is satisfied, the integration formula F = {*Order.needNum* = ∑ *Requirement. Material-Num*}. The operator used for triggering the *Pull Integration* is defined as follows:

```
<extendOpSet>
  <extendOp opID="ExtendedOP"
     opName="import from Workshop Material Requirement ">
  <whereRun BOId=”Order”></whereRun>
  <! —Pull Integration-->
  <function> Order.needNum = sum (Requirement.MaterialNum)
  </function>
```

```
    <condition> Order. needNum, Order. time
</condition>
        …
    </extendOp>
  </extendOpSet>
```

5. Related Works

Several researchers have done work on developing enterprise software based on business object. Typical applications are requirement engineering, software design and implementation. Generally, the business object is used for implementation of business logic, which is composed with appropriate presentation object or web application to become an integrative application for the terminal clients. However, with the improvement of the enterprise software, business object is considered important object type. The business object type, technical object type and application object type are three types of objects based on the principle of separation of concern [26]. And then the business objects are specialized to common business objects, industry specific business objects, company specific business objects and user specific business objects according to the individual requirements of the different roles [27]. Therefore, the business objects are defined as components of the enterprise software that directly represent the business model. The largest granularity of business objects are cooperative business object (CBO) [28]. The CBO is considered the end product of the development process and cooperates with other objects to perform some desired task.

The different business objects have different application scenarios, therefore the characteristics of business object are diversity. In **Table 1** we give the comparisons between our business object and the common BO from OMG and the CBO from O Sims [28]. On one hand, the BO in this paper is lower level than the OMG's BO. On the other hand, the BO in this paper, which is the composition of many fine-grained objects, is one kind of the smaller scale CBO. These fine-grained objects always cooperate partially to meet the requirement in the range of business documents. The software development based on the different BOs is also comparatively diversity. We compared our BOSD to the business process-based methodology which is proposed by Somjit and Dentcho [14] under the background of **Figure 1**. The number of BOs in **Figure 1** is twenty, including eight BOs, eight auditing objects, four association objects and four integration BORs; the number of BO is eight in the method proposed by Somjit. Although they are coarse-grained objects, the reconfiguration cost of BOSD is mostly half of the other method. We only modify the information of the BORs to meet the processes changes.

According to the comparison above mention, our business objects are closest to the component-based implementation for enterprise software. A benefit of our approach is that it defines clearly the relationships between the business objects. The model for software design which is composed of business objects and the relationships is easily transformed on the MDA platform.

6. Conclusions

In this paper, aiming at the problem that current flexible software development methods lack of the systemic methodology and technology support, we present an approach based on coarse-grained business object. By analyzing the changeability of business processes and stability of business objects, we abstract an independent business object as the unit of development and reconfiguration. The three relationships among business objects are defined to describe the variable business processes. Thus, the business objects are assembled to system through their relationships. The implementation of this approach

Table 1. Comparisons between the BOs in literatures.

	OMG's BO	BOSD's BO	Sims's CBO
Counterpart in real enterprise	concept	Business document	The integration application for the user
Software lifecycle	All phases	design phase	All phases
Abstract degree	High	Low	Low
Architecture	undefined	defined	open
Independence	—	Y	Y
Granularity	—	large	larger
Reusable	Y	Y	Y
Easy to program	—	N	N

is supported by the ICEMDA platform.

In conclusion, there are several innovations in the method presented in this paper, as follows: 1) analyze the variable features of the enterprise system, and find out the flexible software can be composed of the changeable business processes and stabile business objects; 2) present an explicit definition of coarse-grained business object; 3) make clear the typical relationships between the business objects; 4) present an method for the flexible software development based on the stable business object, which is implemented by the MDA.

Future works includes: further research on the other relationships between business objects, automatic identification from the business processes, complete the automatic transformation from the PIM to PSM on the platform ICEMDA, etc.

7. Acknowledgements

This work is supported by State 863 High-Tech Program (No. 2009AA04Z153) and State Natural Science Foundation (No. 60773064) of China. The author would like to thank reviewer's helpful suggestions for revising this paper.

REFERENCES

[1] J. Kramer and J. Magee, "The Evolving Philosophers Problem: Dynamic Change Management," *IEEE Transactions on Software Engineering*, Vol. 16, No. 11, 1990, pp. 1293-1306.

[2] W. J. Kettinger and V. Grover, "Special Section: Toward a Theory of Business Process Change Management," *Journal of Management Information Systems*, Vol. 12, No. 1, 1995, pp. 9-30.

[3] D. Kim, M. Kim and H. Kim, "Dynamic Business Process Management Based on Process Change Patterns," *Proceedings of the* 2007 *International Conference on Convergence Information Technology*, Gyeongju, 2007, pp. 1154- 1161.

[4] P. Soffer, B. Golany and D. Dori, "Aligning an ERP System with Enterprise Requirements: An Object-Process Based Approach," *Computers in Industry*, Vol. 56, No. 6, 2005, pp. 639-662.

[5] G. Pour, "Moving toward Component-Based Software Development Approach," *Proceedings of the* 27*th Technology of Object-Oriented Languages* (*TOOLS* 27), Beijing, China, 1998, pp. 296-300.

[6] J. Kotlarsky, I. Oshri, K. Kumar and J. van Hillegersberg, "Towards Agility in Design in Global Component-Based Development," *Communications of the ACM*, Vol. 51, No. 9, 2008, pp. 123-127.

[7] Architecture Board ORMSC, "Model Driven Architecture (MDA)," OMG document number ormsc/2001-07-01. 2001. http://www.omg.org/cgibin/doc?ormsc/2001-07-01.pdf.

[8] E. Breton and J. Bézivin, "Model Driven Process Engineering," *Proceedings of the* 25*th Annual International Computer Software and Applications Conference* (*COMPSAC* 25*th*), Chicago, Illiois, 2001, pp. 225-230.

[9] J. Koehler, R. Hauser, S. Kapoor, F. Y. Wu and S. Kumaran, "A Model-Driven Transformation Method," *Proceedings of the* 7*th Enterprise Distributed Object Computing Conference*, Brisbane, Queensland Australia, 16-19 September 2003, pp. 186- 197.

[10] M. P. Gervais, "Towards an MDA-Oriented Methodology," *Proceedings of the* 26*th Annual International Computer Software and Applications Conference* (*COMPSAC* 2002), England, 26-29 August 2003, pp. 265-270.

[11] M. Keith, H. Demirkan and M. Goul, "Service-Oriented Software Development," *Proceedings of the* 2009 *Americas Conference on Information Systems* (*AMCIS* 2009), 2009, America, p. 100.

[12] P. Wohed, W. Vander Aalst, M. Dumas and A. Ter, "On the Suitability of BPMN for Business Process Modeling," *Business Process Management*, Vol. 4102, 2006, pp. 161-176.

[13] D. Hollingsworth, "The Workflow Reference Model," Technical Report, Workflow Management Coalition, TC00-1003, December 1994.

[14] A. Somjit and B. Dentcho, "Development of Industrial Information Systems on the Web Using Business Components," *Computers in Industry*, Vol. 50, No. 2, 2003, pp. 231-250.

[15] S. Arbaoui, J. C. Derniame, F. Oquendo and H. Verjus, "A Comparative Review of Process-Centered Software Engineering Environments," *Annals of Software Engineering*, Vol. 14, No. 1-4, 2002, pp. 311-340.

[16] R. Sangwan, C. Neill, M. Bass and Z. E. I. Houda, "Integrating a Software Architecture-Centric Method into Object-Oriented Analysis and Design," *Journal of Systems and Software*, Vol. 81, No. 5, 2008, pp. 727-746.

[17] I. Reinhartz-Berger, D. Dori and S. Katz, "OPM/Web-Object-Process Methodology for Developing Web Applications," *Annals of Software Engineering*, Vol. 13, No. 1-4, 2002, pp. 141-161.

[18] H. Liu and D. P. Gluch, "Conceptual Modeling with the Object-Process Methodology in Software Architecture," *Journal of Computing Sciences in Colleges*, Vol. 19, No. 3, 2004, pp. 10-21.

[19] W. Kozaczynski, "Architecture Framework for Business Component," *Proceedings of the* 5*th International Conference on Software Reuse*, Canada, 2-5 June 1998, pp. 300-307.

[20] A. Karakaxas, B. Karakostas and V. Zografos, "A Business Object Oriented Layered Enterprise Architecture," *Proceedings of the* 11*th International Workshop on Database and Expert Systems Applications*, Greenwich, London, U.K., 4-8 September 2000, pp. 807-810

[21] M. Tsagias and B. Kitchenham, "An Evaluation of the Business Object Approach to Software Development," *Journal of Systems and Software*, Vol. 52, No. 2-3, 2000, pp. 149-156.

[22] L. Nie, X. Xu, C. David, Z. Gregory and D. Zhan, "GRAI-ICE Model Driven Interoperability Architecture for Developing Interoperable ESA," *The International Conference on Interoperability for Enterprise Software and Applications*, United Kingdom, 12-15 April 2010, pp. 111-121.

[23] D. Zhan, J. Feng, L. Nie and X. Xu, "ICEMDA: An Interoperable Configurable Executable Model Driven Architecture," *Chinese Journal of Electronics*, Vol. 36, No. 12A, 2008, pp. 120-127.

[24] J. Li, D. Zhan, L. Nie and X. Xu, "Design and Implementation of a MOF Based Enterprise Modeling Tool," I-ESA, China, 2009.

[25] J. Feng, D. Zhan, L. Nie and X. Xu, "Pattern Based Code Generation Method for the Business Component," *Chinese Journal of Electronics*, Vol. 36, No. 12A, 2008, pp. 19-24.

[26] R. Shelton, "Business Objects Response from Open Engineering Inc. to OMG BODTF RFI-1," http://www. OMG.com/OMG Document bom/97-10-07

[27] C. Casanave, "Business-Object Architectures and Standards," *Proceedings of the* 10*th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications* (*OOPSLA'*95), USA, 1995.

[28] O. Sims, "Business Objects: Delivering Cooperative Objects for Client-Server," *The IBM McGraw-Hill Series McGraw-Hill Book Company*, 1994.