A Log-based and Traceability-Oriented Business Object Association Model for Code Generation

Zhongjie Wang, Dechen Zhan, Xiaofei Xu

School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China

Summary
Building well-designed business models is a key step to implement agilely reconfigurable enterprise software and applications (ESA) to adapt to rapid changes in business environments. Traditional business modeling methods can not effectively deal with complex association relationships between business objects; therefore it has great influences on extendibility, adaptability, second-round development efficiency and traceability of ESA. To solve this limitation, in this paper we present a log-based and traceability-oriented business object association model, in which complex and volatile associations between objects are clearly separated from the inner structure of objects and three numerical association styles, i.e., 1 to 1, 1 to *n* and *n* to 1, are emphatically discussed. Business logics on objects are classified into two types of operations, i.e., simple operations that deal with inner logics in a simple object, and complex operations which deal with numerical associations between objects. Run-time states of business objects and association information between objects are separated as logs for further traceability.

Key words:
Business models, business objects, association, logs

1. Introduction

Modern enterprises are in a rapidly and violently changing business environment, which leads to frequent changes on management patterns and business processes of enterprises. To support such changes, broadly applied enterprise software and applications (ESA) must have the capacity of agile reconfiguration[1] .

Building well-designed business models are considered as a key and prior activity to support agility of ESA. Although core of business models is the process view, each business process may be elaborately decomposed into three parts, i.e., a set of business objects to be manipulated (e.g., bills, resources, humans, reports, etc), a set of operations that take effects on these business objects, and then, a set of rules that describes the execution or triggering sequences of these operations. Therefore we may draw a conclusion that only if business object and the corresponding operation models are flexible enough, may the agility of ESA be ensured.

Traditional ESA business models are usually constructed with process-oriented way, i.e., modeling each business as a process and a set of activities. This approach has some obvious shortcomings, e.g., (1) if a modeler has not yet found all the business process in an enterprise, then the process models would be incomplete; (2) structures of some processes are comparatively complicated, therefore the quality and soundness of the model will be determined to a large extend by the ability and experience of the modeler, etc. If we start with another view, i.e., business object modeling, above limitations will be eliminated. This is because number of business objects in an enterprise is determinate and numerable, therefore if all the business objects are identified and their attributes, states, operations are obtained, and then completeness and soundness of business models are easily ensured.

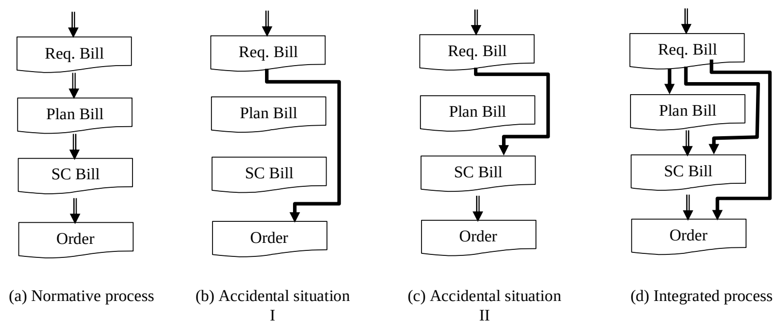

Fig. 1 An example of associations between different business objects

Manuscript received March 25, 2006.
Manuscript revised March 30 , 2006.

Business objects do not isolatedly exist, and there are various associations between objects. Fig. 1 shows a simple example in *Procurement* domain of enterprises, e.g., Fig. 1 (a) describes the business “Gather/decompose a set of procurement requirement bills (*Req. Bill*) to draw the procurement plans (*Plan Bill*), then select proper suppliers according to the results of comparison on quality and price (*SC bill*), finally create the procurement orders (*Order*) with the chosen providers.” There are four business objects and three associations between them.

In object models, elements that usually change are mainly the associations, and the inner attributes of a business objects seldom changes. For example, in routine situations, an enterprise may do its procurement process with the style of Fig. 1(a), however in some exceptive circumstances (e.g., urgent procurement requirements), the final procurement orders may directly created from *req. bill*s without planning, just as shown in Fig. 1(b)(c). Therefore, an ESA must have the ability to simultaneously support all possible situations (e.g., Fig. 1(d)).

In order to produce the final executing source codes according to business models, besides basic business logics in each object, those associations between objects must also be elaborately coded. One of the most familiar strategies is to design an association as one or several attributes in related objects, and when relationships between object instances occur at run-time, values of these attributes are written in. In such situation, the attribute set of each object will contain a mass of redundant association attributes (e.g., the attribute “*quantity of planed procurement requirements*” in *req. bill* expresses the association with *plan bill*) besides self-related information.

This way has very little to be recommended with the reasons that:

(1) Cannot support “*n* to *n*” associations, e.g., a *req. bill* may be planed in multiple *plan bills*, while a *plan bill* may inversely satisfy multiple *req. bill*, therefore it will not realize good traceability.

(2) Increase the complexity of programming and deteriorate agility, e.g., in design phase of ESA, it is impossible to clearly know all the possible associations, therefore it is not possible to design the complete association attribute set for each object, and when the product is applied in practice, when such associations change, a mass of database schemas (e.g., *tables*, *views*, *foreign keys*, *index*, *triggers*, etc) and source codes must be modified.

(3) Increase the chaos of object structure, e.g., with the increasing of variable associations, the number of association attributes in an object will possibly increase linearly, therefore increase the difficulty to effective maintenance.

Consider from the structure of business object models, there are two possible parts, i.e., elements that seldom change and elements that frequently change. The former mainly point at those basic operations on an independent object (e.g., *Create*, *Delete*, *Update*, *Query*, etc) and have no effects on other objects. Every object contains such operations which are stable enough that seldom change along with changing management patterns. The latter mainly refers to associations between objects, which express the relationships and data flows between different business, and by which traceability between business data may be accomplished.

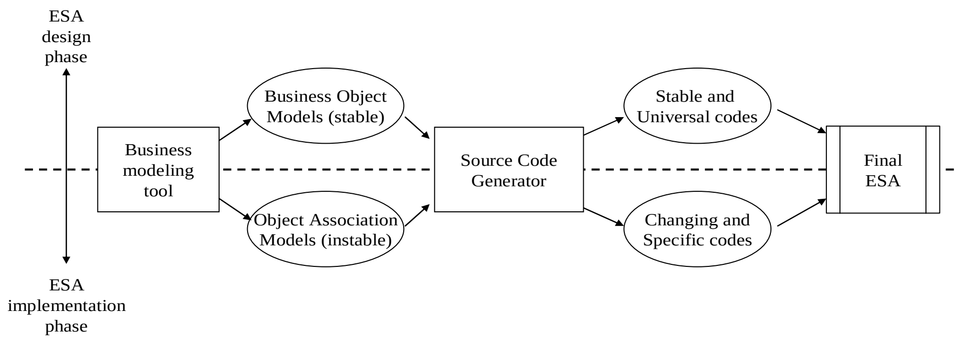
ESA design phase
Business modeling tool
Business Object Models (stable)
Object Association Models (instable)
Source Code Generator
Stable and Universal codes
Changing and Specific codes
Final ESA
ESA implementation phase

Fig. 2 Development process based on business object association models

If we consider the principle “*Separation of Concerns*”, i.e., clearly separating stable business objects from unstable associations, then the two parts may be modeled and coded independently (e.g., the former is mainly done during design phase and the latter is mainly dealt with during implementation phase according to specific requirements) and are then composed together to form the integrated ESA. This way is briefly shown in Fig. 2.

The main purpose of this paper is to present an association/state log based business object association model. Under the help of a code generation tool, this kind of models may be automatically transformed into executable source codes, and during run-time, association and traceability between object instances will be flexibly and rapidly supported.

Rest of this paper is organized as follows. In section 2 we put forward some related works in literatures. In section 3, the log-based object association model are presented in details. In section 4, the detailed development pattern and process based on our model is briefly discussed. Finally is the conclusion.

2 Related works

Association is an important aspect in *Object-Oriented modeling*, and because of the characteristics of frequent changing[2], it is also considered as a puzzle in OO researches [3][4][5].

One of a key principles in software engineering domain is "*Separation of Concerns*"[6][7] with the purpose of improving flexibility of models and systems. This principle is initially derived from the idea of "*separate data structure from program*", and in business process modeling there also appear some similar principles, e.g., "*separate business process from program*", "*separate business rules from business process*", etc[1][6][8]. In a word, "*separate those unstable elements from those stable ones*"[9][10]. For example in [11], aiming at business variations, a model-driven and process-configuration based ESA development style is presented. The author of [2] also considered that associations between business are easy to change, therefore he separated these associations out and presented the definition of "*Rule Object*" with 22 typical patterns (i.e., how to compose business objects and related rule objects). In [12], a new modeling approach of how to separating business object models from business process models is elaborately discussed.

However, these methods mainly focus on associations between macro, process-centered business elements (e.g., activities), and there lacks of enough guidance for applying them to business object associations.

Since association is a key factor to deteriorate flexibility of ESA, "*Simplify associations between elements*" is an obvious approach in ESA modeling [6][7]. However, associations cannot be completely eliminated whatever happens, and for this reason, in literatures there are a lot of methods on how to design *good* associations, such as *Open-Close Principle* (*OCP*), *Acyclic Dependence Principle* (*ADP*), *Stability Dependence Principle* (*SDP*), etc[13]. Other approaches conclude *extended Entity-Relation diagram*[14], *complex association approach* [15], etc. One of the common features of these methods is: first defining object interfaces, then manually coding objects and their interfaces, finally control association behaviors between objects by complex calling methods [5].

Further, there appeared a classical solution for object association in [16], namely, *Object Association Pattern*, in which each association is considered as an independent class and this class is uniformly treated as general objects. In [17], based on the analysis of association's three forms (i.e., 1 to 1, 1 to *n* and *n* to 1), the author presented three methods to implement associations (i.e., *pointer-based*, *matrix-based* and *association class based* approaches), and association class was also regarded as the best one.

Following list some other typical strategies in literatures on this problem.

In [18], *Ontology* is imported to build *Business Object Model* (*BOMs*), and associations are also expressed as ontology. However it is only conceptual modeling and is too coarse to be applied in ESA design.

In [5], *object hook/flange* and *object assembling* were presented. The former implements a dynamic, direct and rapid mechanism for object behavior associations, i.e., all the "*flanges*" are realized in run-time instead of build-time. The latter is based on *object hook interfaces* (rather than the ordinary object interfaces) to achieve domain-oriented dynamic manipulation of object association and automatic maintenance of association semantics.

In [19] the authors presented an *Agent–Object-Relationship* (*AOR*) modeling approach, in which E-R diagram and UML are extended to express some special association types. But AOR did not mention how to implement these associations.

In [20], *Active Business Objects* (*ABOs*) are used to depict the constraints, trigger relations and association operations between objects, with the purpose of message transformation between objects (which is a type of association, too).

In [3], aiming at the uncertainty of business object attributes and associations, the authors transformed the persistent business objects into the generalized table structure and generalized recursion structure, and variable business associations are designed from these abstract data models, i.e., an abstract business object is firstly constructed and bound dynamically with the concrete physical storage structure lately in the deployment phase.

3 Log-based business object association model

3.1 Elements in the model

First we present the basic structure of our model, as shown in Fig. 3. Our main contribution is to extend the traditional business object model[7], i.e., adding Association, State Logs, and Association Log into it, and classifying business operations into simple and complex operations.

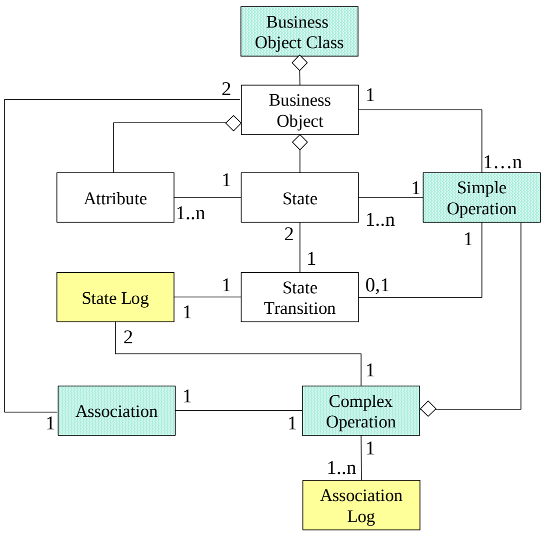

Fig. 3 Basic elements in business object association models

3.2 Business object

Various bills, reports, resources (humans, equipments, fields) in enterprises are jointly called Business Objects in ESA, and may be denoted as:

BO := <*ID*, *name*, *Category*, *Attribute_set*, *State_set*, *StateTransition_set*, *SubBO_set*>

In which,

- *ID* and *Name* are the unique identifier and name of the object;
- *Category* refers to the object class that this object belongs to;
- *Attribute_set* is the attribute set contained in the object, and each attribute is defined as *Attribute* := <*attribute_id*, *attribute_name*, *data_type*> with specific ID, name and data type;
- *State_set* is the state set of the object, and each state is defined as *State* :=<*state_id*, *state_name*, *state_logic*> with specific ID and name. *State_logic* is a condition expression composed of attributes in *Attribute_set* to show when the object is in the state.
- *StateTransition_set* is the state transition set of the object, and each transition is defined as *StateTransition* := <*from_state*, *to_state*> to describe that the transform from *from_state* to *to_state* is allowed.
- *SubBO_set* is the child object set contained in the object. Generally speaking, each object may have zone, one or multiple child objects. $\forall BO$, $\exists BO_1$ makes $BO \in SubBO\_set(BO_1)$, then $State\_set(BO)=\varnothing$, $StateTransition\_set(BO)=\varnothing$.

3.3 Business object class

Those business objects that have the same goals and similar attributes/states together constitute a business object class. For instance, a "*req. bill*" may be classified into "*material req. bill*", "*labor insurance article req. bill*", "*equipment accessory req. bill*", etc. The reason to present this concept is that, objects belonging to the same class may have a majority of similar features, and we may use the class to depict these similarities without modeling them repeatedly.

3.4 Business object state log

During the run-time of ESA, state of each object instance must be efficiently maintained. According to query the state information, operations that may be executed in next step on this object instance will be determined. Current approaches usually add an attribute in the *Attribute_set* of each object to save its state information.

The deficiency of this approach is obvious, i.e., the state of an object at one time is not unique (e.g., "*req. bill*" object may be in "*part planned*" and "*part ordered*" simultaneously) due to concurrent associations with two or more other objects, and one attribute cannot save multiple state information at the same time; if multiple state attributes are adopted, it will then result in redundancy.

For this reason, in our model we separate the state information from a business object itself. State log is adopted to save the state of all the business object instances and states are not related to the attributes any longer.

During run-time, after each operation, the state of each impacted object instance will be written into the state log. By querying this log we may obtain current state of each object instance. Table 1 shows a segment of the log.

Table 1 A segment of state log

Object	Object Instance	Operation	State	Date
Req. Bill	SO0201P001	*Create*	*New*	06-02-01
Order	PO0201P023	*Audit*	*Auditing*	06-02-01
…	…	…	…	…
Req. Bill	SO0201P001	*Audit*	*Auditing*	06-02-02
Req. Bill	SO0201P001	*Audit*	*Audited*	06-02-05
Req. Bill	SO0201P001	*Planning*	*Part planned*	06-02-05
Req. Bill	SO0201P001	*Planning*	*Total planned*	06-02-08
Req. Bill	SO0201P001	*Payment*	*Part paid*	06-03-11
Req. Bill	SO0201P001	*Payment*	*Total paid*	06-03-14
…	…	…	…	…

3.5 Business object association

In ESA, associations between business objects are briefly classified into two types, i.e.,

- *Key association*, which is the most familiar association in OO, e.g., the object "*order*" is key associated with object "*customer*" by the foreign key "*CustomerID*" (in "*order*") and the primary key "*CustomerID*" (in "*customer*").
- *Numerical association*, which describes the "*Create from…*" relationship between objects, e.g., "*req. bill*" is numerical associated with "*plan bill*" by the numerical attribute "*req. quantity*" (in *req. bill*) and "*planned quantity*" (in *plan bill*); or, the "*Allocated to…*" relationship between objects, e.g., "*customer Payment Bill*" is numerical associated with "*Sale Order*" to describe one *payment bill* is paid for what set of *orders*.

Traditional object modeling technique, e.g., *OO*, *E-R diagram*, *IDEF*, usually emphasize on the former associations but ignore the latter ones (they model such associations in process models by *arrows* between two activities), and programmers have to manually control such numerical associations in source code.

As mentioned above, numerical associations are frequently changing, just like Fig. 1(d), there may exist various numerical associations between two objects.

According to the number of object instances of the two parties in one numerical association, it may be further classified into three forms, i.e., 1 to 1, 1 to *n* and *n* to 1 (*n* to *n* may be considered as the combination of 1 to *n* and *n* to 1). Fig. 4 shows a simple example, in which one "*plan bill*" may be simultaneously satisfy multiple "*req. bill*", and vise versa.

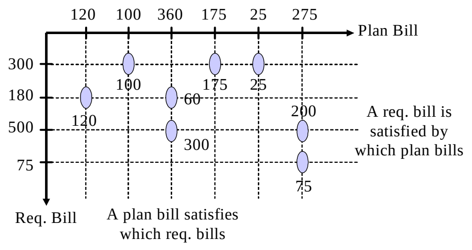
120 100 360 175 25 275
Plan Bill
300
180
500
75
100
175
25
60
120
200
300
75
A req. bill is satisfied by which plan bills
Req. Bill
A plan bill satisfies which req. bills

Fig. 4 *n* to *n* numerical associations between objects

Besides the numerical attributes, a numerical association should have some other information, just as shown in Fig. 5(a), and Fig. 5(b) gives a simple example.

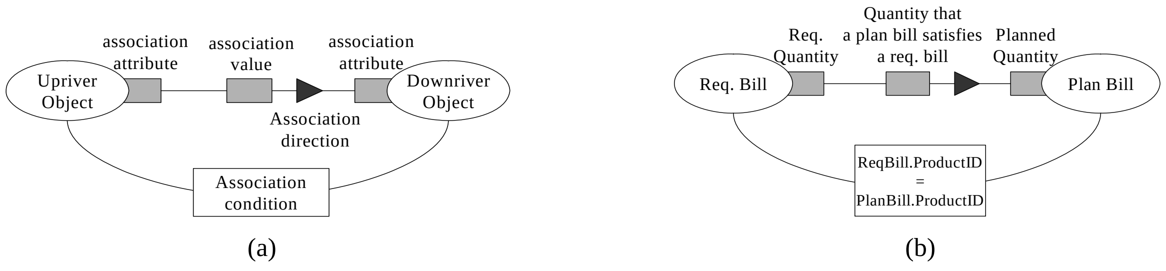

Fig. 5 Numerical association model between objects

3.6 Association log

In order to separate variable parts (associations) from object itself, in our model, we would not save the association information in the attribute set of each object any longer, but distill them to form an independent association log.

Association log actually is a representation of object association model at run-time and records the association between object instances. As long as one complex operation (will be defined in next sub-section) finishes, the concrete association information will be written into the log. By querying the log, flexible traceability between different object instances may be implemented. Table 2 shows a segment of this log (corresponding to the n to n associations in Fig. 4).

Table 2 A segment of association log

Upriver Object	Upriver Obj. Instance	Upriver Asso. Attr./ Quantity	Asso. Value	Downriver. Asso. Attr./ Quantity	Downriver Obj. Instance	Downriver Object
Req. Bill	R001	*Req.Quantity*/300	100	*Planned Quantity*/100	P002	*Plan Bill*
Req. Bill	R001	*Req.Quantity*/300	175	*Planned Quantity*/175	P004	*Plan Bill*
Req. Bill	R001	*Req.Quantity*/300	25	*Planned Quantity*/25	P005	*Plan Bill*
Req. Bill	R002	*Req.Quantity*/180	120	*Planned Quantity*/120	P001	*Plan Bill*
Req. Bill	R002	*Req.Quantity*/180	60	*Planned Quantity*/360	P003	*Plan Bill*
Req. Bill	R003	*Req.Quantity*/500	300	*Planned Quantity*/360	P003	*Plan Bill*
Req. Bill	R003	*Req.Quantity*/500	200	*Planned Quantity*/275	P006	*Plan Bill*
Req. Bill	R004	*Req.Quantity*/75	75	*Planned Quantity*/275	P006	*Plan Bill*
…	…	…		…	…	…

3.7 Simple operation

In our model, operations on business objects are classified into Simple Operation (*SOP*) and Complex Operation (*COP*).

A *SOP* is defined as an atomic operation on a single object, e.g., *Create*, *Read*, *Update*, *Delete* (*CRUD*), etc, and does not related to any other objects. It is also the finest action in business process (cannot be further decomposed) and is considered as the most stable business logic (this is because *SOP* are consequentially existing and does not need to change along with other logics). *SOP* has the following characteristics:

(1) Related to one object;

(2) Make the object occur 0 or 1 state transitions;

(3) May be related to state, i.e., only when the object is in some specific state can it be executed; or may be unrelated to states, i.e., it may be executed at any states;

(4) Objects belonging to the same class may possibly have different SOP sets, e.g., "*Oversea Order*" has a SOP "*Check current exchange rate*", while "*Home Order*" does not have such SOP.

A SOP is formally defined as *SOP* :=<*op_id*, *op_name*, *BO*, *state_related_flag*, *initial_state_set*, *final_state_set*>, in which *BO* is the business object that the *SOP* operates on, *state_related_flag* is a flag to show whether *SOP* is related to states, i.e., *state_related_flag*=1 means that only when *BO* is in one of the states in *initial_state_set* can *SOP* be allowed to execute (e.g., the *SOP* "*Req. Bill Planning*" may only be allowed to execute when the *BO* "*Order*" is in the state "*Audited*"), and after the execution of *SOP*, *BO* should reach the states of *final_state_set*; *state_related_flag*=0 means that *SOP* is not related to states and may be executed at any states (e.g., "*Create New Order*", "*Query Order Information*", etc).

A *SOP* should complete the following tasks:

(1) Business logics (e.g., *CRUD*) on *BO*;

(2) Write *BO*'s state information (after execution of *SOP*) into the state log.

3.8 Complex operation

A complex operation (*COP*) is an operation dealing with an association between objects and come down to two objects in the association (called *Upriver* and *Downriver* objects), denoted as *COP* := <*op_id*, *op_name*, *association_id*, *type*>, in which *association_id* refers to the association that *COP* operates on, *type* represents one of the following:

(1) *Push*: *SOP* is triggered/executed by the upriver object;

(2) *Pull*: *SOP* is triggered/executed by the downriver object;

(3) *Push/Pull*: *SOP* may be triggered/executed by the upriver or downriver objects arbitrarily.

For example, a *COP* related to the association between "*req. bill*" and "*plan bill*" is "*Generating procurement plans from requirements*", which is a pull *SOP*, i.e., this operation should be called in the interface of "*plan bill*" to generate new *plan bill*s according to the data of selected req. bill.

The codes of a *COP* should complete the following tasks:

(1) Query upriver objects instances;

(2) Generate new downriver objects according to the association value (by user's input), e.g., the planned quantity in Fig. 5(b);

(3) Write the association information into association log;

(4) Write the state information of related object instances into state log.

4 ESA development pattern based on business object association model

One of the primary goals of our model is to improve the agility of ESA, i.e., when associations between objects change frequently, ESA may adapt to such changes with high efficiency and low scales of code modifications.

In Fig. 2 we have presented the basic idea of our development process based on our model, and Fig. 6 shows the detailed process.

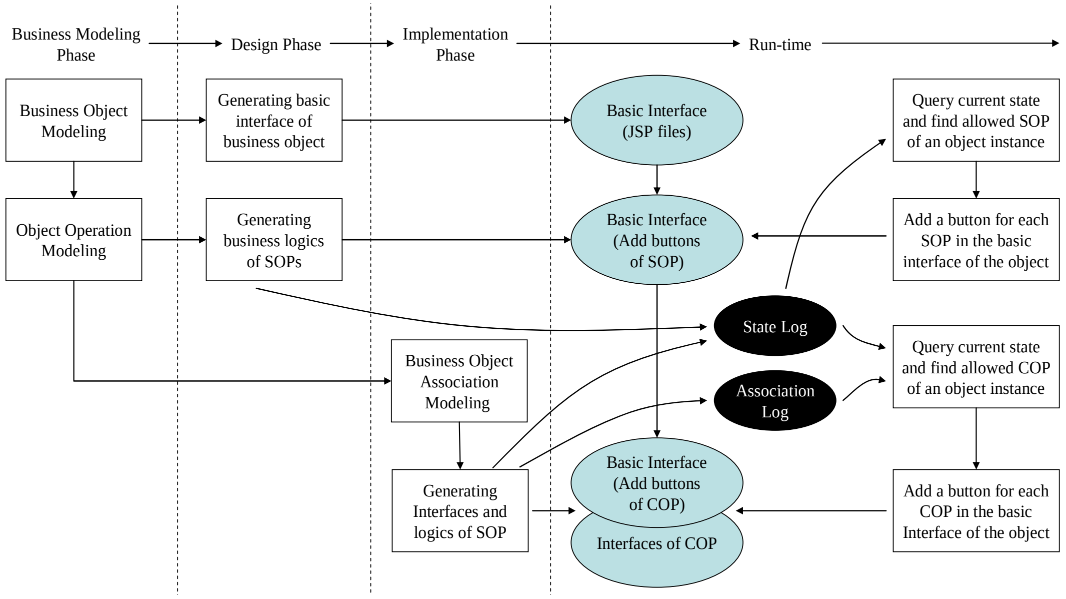
Business Modeling Phase
Design Phase
Implementation Phase
Run-time
Business Object Modeling
Generating basic interface of business object
Basic Interface (JSP files)
Query current state and find allowed SOP of an object instance
Object Operation Modeling
Generating business logics of SOPs
Basic Interface (Add buttons of SOP)
Add a button for each SOP in the basic interface of the object
State Log
Query current state and find allowed COP of an object instance
Business Object Association Modeling
Association Log
Generating Interfaces and logics of SOP
Basic Interface (Add buttons of COP)
Interfaces of COP
Add a button for each COP in the basic Interface of the object

Fig. 6 ESA development process based on business object association model (refined)

Based on the model and process, we have designed a code generation tool to support *automatic code generation*. The input of this tool is the association model, and the output is executable source codes (we adopt *J2EE*-based software architecture, in which user interfaces are *JSP* files, background business logic are *EJB* files, and *MVC* structure is used to compose these files together. The final system is running on *Weblogic* application server platform).

5 Conclusions

Our approach (model and tool) has been applied in several *Enterprise Resource Planning* (*ERP*) projects and has gained significant results, which is mainly reflected on the improvement of developing efficiency and code quality. According to the statistical data, the average second-round developing period of each sub-system (average 10-15 business objects) of ERP is reduced from 1 month to 12 days, i.e., the developing efficiency increased by 60%.

Our approach still has some insufficiencies that need further researches, e.g.,

(1) Because of complexity of enterprise business, state logic and operation's business logic should be formally defined in models, therefore the code generation tool make directly transform them into source codes;

(2) Our approach could only deal with *bill* and *resource* objects; for *report* objects, because their data are from multiple data sources and the relationships between these data are comparatively complex, therefore we temporarily cannot treat with such objects;

(3) Our approach only aims at *numerical* association, but for other types of associations we have not considered them yet.

Acknowledgments

Research works in this paper are partial supported by *the Specialized Research Fund for the Doctoral Program of Higher Education* (*SRFDP*) in China (Grant No. 20030213027) and *the National Natural Science Foundation* (*NSF*) of China (Grant No. 60573086).

References

[1] BAJEC M, KRISPER M. A methodology and tool support for managing business rules in organisations. Information Systems, 2005, 30(6): 243~443.

[2] ARSANJANI A. Rule Object 2001: A pattern language for pluggable and adaptive business rule construction. Proceedings of 8th Conference on Pattern Languages of Programs (PLoP2001). Monticello, Illinois, USA, 2001. 1~33.

[3] BIAN S H, XUE J S, SONG X Y. Dynamic persistent object in ERP System. Computer Integrated Manufacturing Systems, 2003, 9(5): 378~383.

[4] LIANG Y. Establishing the framework for business object analysis and design models. Proceedings of the 6th International Conference on Object Oriented Information Systems. 2000. 155~162.

[5] WAN J C, LIU S. Object assembling and automatic maintenance of its association semantics. Journal of Software, 2002, 13(5): 1013~1017.

[6] WANG Z J, XU X F, ZHAN D C. Reconfiguration oriented business models for enterprise information system. Computer Applications, 2005, 25(8): 1861~1864.

[7] WANG Z J, XU X F, ZHAN D C. RO- BPM: A reconfiguration-oriented business process model. Computer Integrated Manufacturing Systems, 2004, 11(11): 1349~1355.

[8] HARMON P. Managing business processes. Business Process Trends, 2003, 1(8): 1~16.

[9] HAMZA H S. SODA: A stability-oriented domain analysis method. Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. Vancouver, Canada: ACM Press, 2004. 220~221.

[10] WANG Z J, XU X F, ZHAN D C. Component Granularity Optimization Design Based on Business Model Stability Evaluation . Chinese Journal of Computers. 2006, 29(2): 239-248.

[11] JIN J W, JIN Ye. Research on key technologies of ERP based on model-driven & process-configuration. Computer Integrated Manufacturing Systems, 2005, 11(7): 986~995.

[12] SNOECK M. Separating business process aspects from business object behaviour. New Directions in Software Engineering. Leuven University Press, 2001.

[13] MARTIN R C. Agile software development: principles, patterns, and practices . New York: Prentice Hall, 2002.

[14] SAIEDIAN H. An evaluation of extended entity-relationship model. Information and Software Technology. 1997,39(7):449~462.

[15] KRISTENSEN B B. Complex associations : abstractions in object-oriented modeling. Proceedings of the 9th Annual Conference on Object-Oriented Systems, Languages & Applications(OOPSLA'94). Portland, Oregon: ACM, 1994. 272~286.

[16] BOYD L. Business patterns of association objects. In Martin, R.C.; Riehle, D.; Buschmann, F. (eds) Pattern Languages of Program Design 3, Addison-Wesley, 1998, 395~408.

[17] LIU Z Z. Implementation of association in OO modeling. Computer Engineering, 1997, 23(6): 38~40 +52.

[18] HAMZA H S. Developing business object models with patterns and ontologies. Proceedings of 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2005. 106~107.

[19] WAGNER G. The Agent–Object-Relationship metamodel: towards a unified view of state and behavior. Information Systems. 2003, 28(5): 475~504.

[20] Li H F, SU Y W. Business object modeling, validation, and mediation for integrating heterogeneous application systems. Journal of Systems Integration. 2001, 10(4): 307~328.

Zhongjie Wang received his B.S., M.S. and Ph.D. degrees in computer science from Harbin Institute of Technology in 2000, 2002 and 2005 respectively. He is now a lecture in computer science in School of Computer Science and Technology at Harbin Institute of Technology (HIT), PRC. He His research interest includes software engineering, software reuse, software reconfiguration, software component related techniques.