JBOORET: an Automated Tool to Recover OO Design and Source Models

Hong Mei, Tao Xie, Fuqing Yang
Department of Computer Science & Technology, Peking University, Beijing, China 100871
Email: {meih, taoxie, yang}@cs.pku.edu.cn

Abstract

This paper introduces a reverse engineering tool, JBOORET (Jade Bird Object-Oriented Reverse Engineering Tool). This tool is developed by adopting a parser-based approach to assist the activity of extracting the higher-level design and source models from system artifacts. A conceptual model is formulated as the knowledge representation. Multi-perspective design and source models are recovered by JBOORET based on the comprehensive program information extracted from source code. Its flexible user interface can assist users to browse the detailed information of design and source models by using the selection and compaction mechanism. This paper discusses the design principles and decisions of JBOORET and describes its implementation.

Keywords: Reverse Engineering, CASE Tool, Object-Orientation, Program Understanding

1. Introduction

Legacy system comprises a great deal of knowledge of the enterprise, such as system requirements, design decision and business plan. In order to use existing assets in legacy systems effectively, it is important to develop a systematic strategy for continuable evolution of legacy systems. Reverse engineering helps users understand a system by identifying objects in the system, finding out the relations between these objects and expressing the system in an abstract way.

To reduce the time and efforts software personnel spend on legacy system analysis, an automated reverse engineering is required to facilitate the recovery of design and source models and the generation of reports and documents recovered from existing system. A good reverse engineering tool can aid software engineers to analyze and understand the complex legacy software system. It plays an important role in the process of reengineering activities.

There are many reverse engineering tools implemented in software industry and academia. Because many of current legacy systems were written in imperative or procedural languages, a number of those reverse engineering tools are designed for procedural source code, such as Refine/C [1] and Rigi [2]. These tools mainly focus on the recovery of interprocedural dependence, data structure and control/data flow graph etc. However, with the object-oriented technology and object-oriented program languages adopted by software developers, there appear more and more object-oriented software systems in software industry, which has led to the development of reverse engineering tools for object-oriented software. These tools not only inherit some functions of procedural reverse engineering tools, but also realize some new features such as class hierarchies and class nest diagram recovery. However, most of current reverse engineering tools can not produce satisfactory views of the application for users. Some tools may not extract the accurate and complete information from source code and consequently the recovered models from the application are not comprehensive enough to perform forward engineering. Some other tools may not represent the higher-level abstraction and design satisfactorily, especially when the size of the legacy system is rather large. Generally the whole higher-level view of the application is less readable and even unmanageable with the growth of the size.

In our work, we explored some approaches in designing the object-oriented reverse engineering tool in order to address the above problems. With the parser-based approach, Jade Bird Object-Oriented Reverse Engineering Tool (JBOORET) is designed to recover the comprehensive higher-level models from the source code. It also produces multi-perspective models and presents rich information of models to users satisfactorily by selection and compression mechanisms.

The paper is organized as follows. In section 2, we first explore the design principles and decisions of JBOORET. In section 3, we introduce the implementation of our JBOORET. In section 4, we give a brief introduction of a case study. We discuss related work in section 5. Finally we reach a conclusion in section 6.

2. Design Principles and Decisions of JBOORET

Canfora etc. [3] proposed some problems of current reverse engineering tools as follows. ①If tools fail to

0-7695-1372-7/01 $10.00 © 2001 IEEE

identify the right level of details in recovered design and lack proper design presentation strategies; ②Or recover only a small portion of the design information software maintainers need; ③Or are not flexible enough and can't be easily tailored and enriched in the operating environment, it would hinder the wide application of reverse engineering tools.

In our work, we have designed JBOORET for C++. Similar to Stan Jarzabek's [4] work, a model-based approach is adopted to design this tool. A conceptual model of object-oriented program information is defined and stored in physical storage – a relational database. Therefore the variances in requirement of higher-level design and source model will not affect the analysis front end and the low-level conceptual model extracted from the source code. To effectively support the program understanding and software reuse, JBOORET should be designed according to following principles:

1) The recovered models should describe the program from different granularity. The reversed models should represent not only the algorithm and data structure of program (such as the member attributes and member methods), but also the system architecture, describing the relations between each part of the program (such as class structure and component dependency relation etc.).
2) The recovered models should be easily manipulated by users. When users are in the process of understanding the program, the tool should help them manipulate the recovered models and the entities in them.
3) The generated models should be able to be used directly in forward engineering. Software maintenance and software reengineering are the iterative processes between reverse engineering and forward engineering. CASE tool for forward engineering should be able to utilize the recovered models directly.
4) The adopted analysis approach should ensure the correctness and completeness of recovered models to be used in forward engineering. Otherwise the result of the forward engineering based on the recovered models would be inaccurate.

Some specific design principles and decisions are discussed in light of the three phases of the reverse engineering process put as follows.

2.1. Data Analysis

Data analysis is a core part of the reverse engineering tool. Generally there are two approaches in analyzing the source code. One is a lexical approach, which is applied by specifying the regular expressions that are matched to the program source code. The advantages of this approach are its versatility, adaptability and flexibility [5]. However, the lexical approach has its disadvantages in certain circumstances. Sometimes this approach extracts unexpected information or misses the expected information. Additionally it has to scan the whole source code every time it deals with a specific model specification.

The other one is the parser-based approach, which uses the parser to analyze the source code based on the mature compiler technology. It can address those problems that occurred in the lexical approach. The more accurate and comprehensive program information is extracted, the more desirable this approach is for reverse engineering tool. Therefore, the parser-based approach should be adopted to recover the design and source models in JBOORET. By analyzing the design and implementation parts of the application, both the design and source models should be extracted.

Additionally JBOORET should adopt incremental parsing approach to analyze the software system, which is changing during the reverse engineering process. Because it only analyzes those parts, which have changed since the latest analysis, it reduces the parse time and resource. Moreover the incremental parsing technology would effectively support the incremental reengineering process.

2.2. Knowledge Organization

There are several types of data model to organize the knowledge collected from source code. JBOORET should formulate a conceptual model for object-oriented program. The conceptual model is considered as the low-level program information model. During the parsing process, only the low-level program information model is stored in the physical storage. And the higher-level design and source models are derived from the program information model. Thus, the analysis parts and data models become independent from many variances in requirements of different reverse engineering tasks. The tool should not build all higher-level design and source models in advance, but extract those selected higher-level models and construct representations on users' demand. Meanwhile, there is no need to construct different analysis front ends for various higher-level model extractors for desired models. By sharing the common conceptual model, it needn't parse the source code every time when it wants to produce one desired model. It is because the information in the conceptual model, obtained by analyzing the source code only once, can be used to produce different higher-level models many times.

It is a general practice to store the content of the conceptual model in database. And because the conceptual model of program information is separated from the physical storage, it enables the tool to be easily adapted to different database platforms. To support various design and source models for different requirements, the conceptual model should be

comprehensive and well defined.

2.3. Information Presentation

In this phase, the tool presents the higher-level design or source models to users. The functions of the model representations have a great impact on the effectiveness of applying the reverse engineering tool. To reduce the time and efforts to perform the task of model representations, a mechanism should be designed to construct the higher-level design or source models on demand, not in advance. And the tool should also cache those frequently browsed models, which are not too large but are relatively too expensive to be derived [6].

The tool should provide different levels or perspectives of design or source models for users. Then users can select the specific level or part of detail information they wish to view. Many current object-oriented reverse engineering tools, without subsystem division, only recover the whole class diagram. Then the complex class diagram recovered from large application is generally less readable and manageable. Therefore it is necessary for our tool to provide different granular models.

Moreover, the tool should export some necessary higher-level models to object-oriented development tools to facilitate forward engineering. And it should also provide a flexible user interface to allow users to interact with the tool by manipulating the entities in the recovered higher-level models. Selection and compression are two fundamental means of reducing information complexity [7]. The tool is desired to support these two means to overcome the scale problems of information space by providing a user interface to compress or expand any composition of its sub-spaces. Finally the tool should support users to switch from the higher-level models to the actual source code and highlight the related part in the source code, so that the users can understand the mapping between different levels easily.

3. Implementation of JBOORET

The JBOORET for C++ consists of three major components: a data extractor, a knowledge manager and an information presenter, as shown in figure 1. This tool analyzes source code of the program statically in the way of incremental parsing, extracts program information according to a conceptual model formed with Enhanced Entity Relationship model, and stores the information in a relational database.

The flexible architecture adopted by JBOORET has many advantages. This approach conforms to the principle of separating data extraction from information presentation, thereby avoiding repeating analysis process for each higher-level model extraction. The separation between data extractor and knowledge manager makes the analysis part front-end independent of other parts, so that when the tool is required to be adapted to support another similar object-oriented program, the adaptation only affects the components of data extractor. Similarly the separation between information presenter and knowledge manager makes the model extraction part independent of other parts in JBOORET, and therefore the changes in the requirements of higher-level models don't affect data extractor and knowledge manager.

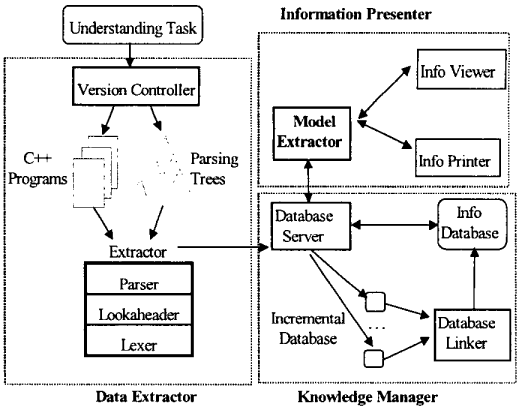

Figure 1. **JBOORET Architecture Overview**

3.1. Data Extractor

The front end analyzes C++ source code, extracts program information and stores it into the program information database through the database server. The model extractor uses the program information via the database server. Only the front end needs to know the exact syntax of the C++ language, which means that when the target language changes, it only affects the front end.

Instead of using standard compiler tool LEX, the lexer (lexical analyzer) is customized for the front end, incorporating a special preprocess into lexical analysis. Receiving physical source code, the customized lexer can extract program information, say, comments and include-file relationship, needed for program analysis. Moreover, it accurately associates C++ program entities, such as class, object, statement, macro, and so on, with their physical location in the source code. To easily update the front end to support analysis of different C++ languages, we implemented the parser around YACC. Normally YACC accepts LALR grammar, but C++ grammar is inherently ambiguous and definitely not LALR. Token lookahead technique is employed to disambiguate the parser generated by YACC.

Incremental parsing is adopted to make it possible to parse only the modified portion during the changes of the source files. JBOORET creates an incremental database for each source file and then links all incremental

database to construct the large program information database.

3.2. Knowledge Manager

JBOORET forms a conceptual model for C++ programs employing Enhanced Entity Relationship (EER) model. According to EER model, C++ programs are viewed as a set of entities and relationships between them, both of which may have a set of attributes. To support various higher-level design and source models for different requirements, the conceptual model should be comprehensive and well defined. The EER model of JBOORET is shown in figure 2, in which the rectangle stands for entity and the rhombus means relationship.

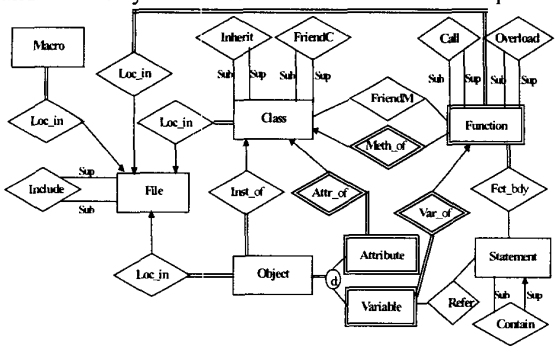

Figure 2. **JBRET Conceptual Model**

JBOORET EER model defines six kinds of entities: Macro, File, Class, Function, Object, and Statement, among which entity Object can be divided into two kinds of sub-entities based on its declaration location: Attribute and Variable. Strong entity is an entity that has a key. For example, entity Macro's name is its key since the name Macro is unique in C++ language. On the other hand weak entity is an entity that has no key. For example, entity functions can have the same name due to function overload and thus they are not unique. In EER model, a strong entity is illustrated as a single-line rectangle, while a weak entity is illustrated as a double-line rectangle.

In this model, relationships exist between entities according to their lexical and semantic relations. For example, in C++ language, classes are always defined in a specific source file, and classes and files have lexical position relation. Therefore, relationship Loc_in exists between entity Class and entity File and its coordination is one-to-many. In EER model, relationship is illustrated as a rhombus, which is connected to two corresponding entities by two lines. Since a weak entity has no key, it must depend on another entity to exist and there is a kind of so-called dependence-relationship between the two entities. The dependence-relationship is represented by a double-line rhombus in EER model.

3.3. Information Presenter

In the information presenter component, a basic symbol information table in memory is constructed from a low-level program model every time users open an analyzed program project to browse. This basic symbol information table comprises the frequently used information of each symbol in memory, which includes the class name, attributes of the class, methods of the class and etc. To improve the access speed to the symbol information table, we organize it as a harsh table. Other complementary information infrequently used, such as the reference information, physical position information, is loaded only on user's demand during the interactive manipulation process. To reduce the time to access the program information database during the interactive process and thus to increase the access speed, a circular cache with a limited capacity is constructed in memory to store the most recently used complementary information. To be simply implemented, FIFO strategy is adopted to replace the information in cache when its capacity is full. During the presenting process, the source information of the higher-level models is first searched in the symbol information table rather than in low-level conceptual model stored in the relational database to improve the efficiency.

The tool provides two major means to present higher-level design or source models. It can export the models to the Object-Oriented Development Tool to support forward engineering, and at the same time provides flexible user interface for users to manipulate the extracted models.

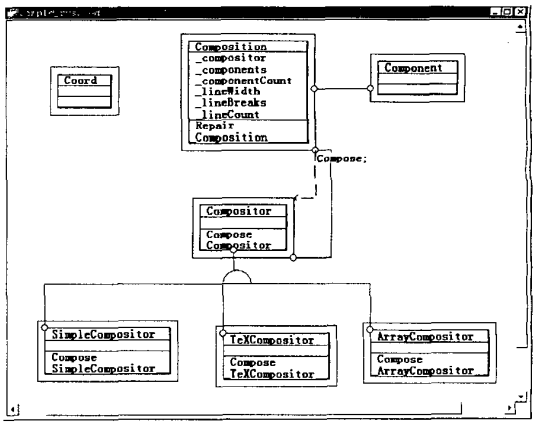

Figure 3. **JBOO class diagram recovered by JBOORET**

JBOORET can export the class diagram recovered from program information database to our Jade Bird Object-Oriented Design Tool (JBOOD). JBOOD is one of the object-oriented development tool set JBOO, which supports object-oriented analysis, design and programming. Reversed JBOOD document describes the

following information of software: classes, classes and objects, attributes, services, classification structure, composition structure, instance link and message link. Figure 3 shows the reversed JBOO design document -- class diagram of sample source code of design pattern Strategy [8].

JBOORET can also export the reversed design model document to currently popular object-oriented development tool Rational Rose 98[1] [9], which includes the following three views:

- Use case view: describes the implementation details of class methods, such as method's message trace diagram, in the forms of sequence diagram and collaboration diagram.
- Logic view: describes the static relation of system, in the forms of package, the dependency relation and class diagram (class, attributes, operations and relations between classes).
- Component view: describes the relations between file components, in the forms of the include relations between files.

Although above recovered models provide an automated way to understand the system and facilitate the forward engineering, the models are usually too large and complex to be easily handled by users with the size of application increasing. Especially when analyzing a large application, the scale of the information in the extracted models is hard to manage if they are not divided into subsystems. However, subsystem decomposition is not easily supported by an individual tool and is insufficiently implemented without manual support. It generally requires application-specific or domain knowledge acquired manually.

To address this problem, JBOORET provides complementary design or source models, which can be manipulated easily by users. Users can search, select, edit or browse the entities in the models to enable users to decide what to focus on and what to ignore. JBOORET adopts multi-perspectives to present the extracted models, including class hierarchy tree, class ancestry tree, class nest tree, file include tree, source code view, program structure graph and cross reference etc.

In order to aid users to focus on the part they are most concerned and temporarily ignore other parts, users can compress the sub-tree in tree models view. The compaction and selection is supported in the complementary models generated by JBOORET. Additionally when users double-click the object in the recovered models, the tool can switch the focus to the actual source code, the part of which is associated with the selected object. At the same time the related part of the source code will be highlighted. Moreover JBOORET supports advanced searching of the symbols, such as class names, method names and variable names, and present the detailed information of some specific symbols to users.

[1] Rational Rose is a trademark of Rational Software Corporation.

4. Case Study

In component-based software development environment, to acquire the reusable components, it is feasible to extract them from existing systems. During the component extraction process, JBOORET is used to understand the legacy systems by providing multi-perspective higher-level models. After the candidate reusable components are identified during the reusable component identification process, JBOORET is applied to support the reengineering of the components by recovering the higher-level models for forward engineering. Additionally before the component users compose the selected components for new application, JBOORET is also adopted to comprehend the components or adapt them to fulfil the new requirements in target systems.

Currently JBOORET is implemented to run on Windows 98/95/NT PC platform, and it is capable of analyzing large C++ program. The tool has already been utilized to recover the design and source models from the 76 sample projects of MS Visual C++ 6.0 (695738 lines of code). The tool has also analyzed the source code of Jade Bird CASE tools (285801 lines of code) and the source code of freeware CORBA product OmniBroker (53435 lines of code). The tool exports the design and source models to JBOOD and Rose 98 and produces complementary multi-perspectives. To compare the results of reverse engineering tools, we perform the same task with the Reverse Engineering Tool built in Rose 98 and JBOORET. Table 1 shows the information of higher-level models recovered by JBOORET and ROSE RET.

Table 1. **The information in Recovered Models generated by JBOORET and ROSE RET**

OOP (Source Code)	JBOOD Model	Rose Model	Rose RET
Class	Class and Object	Class	Y
Class Member Variable	Attribute	Attribute	Y
Class Member Function	Service	Operation	Y
Class Inheritance	General-special Structure	Generalization	Y
Class Embedded Object	Whole-part Structure or Instance Link	Aggregation	Y
Class Embedded Object Pointer	Whole-part Structure or Instance Link	Aggregation	Y
Method Call between Classes	Message Link	Association	N
File Include relation		Component Diagram	N
Single Method Call Train		Scenario Diagram	N

The leftmost column lists the entities in source code, the second lists the corresponding representation entities in JBOOD model and the third one lists those in Rose Model. The rightmost column shows whether the Rose Reverse Engineering Tool (Rose RET) supports corresponding entity recovery or not. In case study, Rose RET is found to be incapable of correctly analyzing the Macro information in the C++ source code and sometimes present some program information from source code incorrectly.

5. Related work

Rigi [2] is a flexible tool for architectural understanding at the University of Victoria. This tool provides a parser to support common imperative programming languages, a repository to store the extracted information and an interactive editor to manipulate program representations. This tool adopts the layered views, ShriMP view and layout algorithms to facilitate the information browsing. And the entities in the recovered models can be easily manipulated. Rigi is programmable through a scripting language and provides a customizable interface. However, the analysis function of this tool is not satisfactory and the generated views are limited mainly to call graphs.

SNIFF+ [10] adopts a fault-tolerant approximate parser to analyze the source code. It can parse those incomplete or syntactically incorrect source codes by performing a partial parse looking for specific syntactic constructs. But it is less desirable in forward engineering support because of the information incompleteness. Moreover, it is not extensible for reverse engineering tasks.

Refine/C [1] is a reverse engineering tool for C programs. It provides a parser to produce the internal representation, an abstract syntax tree which provides the information for model extraction. It also provides a good basis to be extended to specific applications by providing programmable query for users. But its user interface is not flexible and it lacks search function for symbols in application. The manipulation of the recovered model is less satisfactory.

Lightweight lexical source model extraction approach proposed by G. Murphy and D. Notkin [5] adopts a lexical approach to recover the high level models. It allows extracting information from various types of software artifacts. This lightweight approach reduces the effort in developing a flexible and tolerant extractor. However, as is discussed above, the accuracy and completeness of the extracted models are not satisfiable in this approach.

6. Conclusion

We adopted a model-based approach to develop JBOORET. A comprehensive information conceptual model for C++ is formulated, which provides a good basis for higher-level abstraction model extraction. Because we have adopted the parser-based approach to analyze the program, the design and source models recovered by JBOORET are accurate and complete enough to effectively support the reverse engineering, forward engineering and program analysis. It can export design and source models to JBOOD and Rose 98 to support forward engineering conveniently. It also provides complementary multi-perspective models and flexible user interface for users to search, filter and select the program representation. Moreover it assists users to compress or expand the information to facilitate the information focus, especially when the recovered models are rather large and complex.

7. Acknowledgements

This effort is sponsored by the State 9th Five-Year Plan, the State 863 High-Tech Program, and Natural Science Foundation of China. It also got support from Ricoh Company, Ltd., Japan.

8. References

[1] G. Kotik and L. Markosian, "Program transformation: the key to automating software maintenance and re-engineering", Technical Report, Reasoning Systems, Inc., USA, 1999.

[2] Wong, K., Corrie, B. D., Muller, H. A., Storey. M.-A. D., Tilley, S. R., and Whitney, M., Rigi V User's Manual, Department of Computer Science, University of Victoria, Victoria BC, 161 pp.

[3] Canfora, G., Cimitile, A. And de Carlini, U., "A logic-based approach to reverse engineering tools production", IEEE Transactions on Software Engineering, 1992 18(12), 1053-1064.

[4] Stan Jarzabek, Guosheng Wang. "Model-based Design of Reverse Engineering Tools", Software Maintenance: Research and Practice", October, 1998, pp 353-380

[5] Gail C. Murphy, David Notkin, "Lightweight Lexical Source Model Extraction", ACM Transactions on Software Engineering and Methodology, Vol.5, No.3, July 1996, Pages 262-292

[6] D. C. Atkinson, W. G. Griswold, ``The Design of Whole-Program Analysis Tools," Proceedings of the 18th International Conference on Software Engineering, Berlin, IEEE, pp. 16-27, March, 1996.

[7] Javed I. Khan & I. Miyamoto, "Integrating Abstraction Flexibility with Diverse program Perspectives, Proceedings of the 17th Annual International Computer Software and Applications Conference, COMPSAC'93, Phoenix, November 1993, pp. 186-192.

[8] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA: Addison-Wesley, 1995.

[9] Rational Software Corp. Rational Rose 98 [online], Available WWW<URL: http://www.rational.com/rose/>. {1998}

[10] Bischoffberger, W. R., Sniff-A pragmatic approach to a C++ programming environment. In Proceddings of the 1992 Usenix C++ Conference, USENIC Assoc., Berkeley, Calif., 67-81.