C++ Program Information Database for Analysis Tools[1]

Yuan Wanghong, Chen Xiangkui, Xie Tao, Mei Hong, Yang Fuqing
Department of Computer Science and Technology
Peking University, Beijing 100871, P. R. China
E-mail: yuanwh@163.net

Abstract

Program information extracted from source codes is valuable for research in many software engineering fields. Many program analysis tools in these fields usually share some common program information. To support multiple analysis tools based on common program information, it is practical and feasible to store information into database. This paper describes a C++ program information database, which is comprehensive enough to support many analysis tools. To employ the idea of incremental paring, the C++ program information database is linked by multiple incremental databases, which, in turn, are built by extracting information from source codes according to a C++ program conceptual model.

Keyword C++, object orientation, program analysis, incremental parsing, program information database

1. Introduction

Program source codes are usually the primary information source of existing software systems, and are valuable for research in many software engineering fields, such as software testing, software maintenance, reverse engineering, reengineering, and software reuse. To support these research, many code analysis tools have been proposed and reported for different requirements. The following are several actually implemented systems: PUNS[1], developed by IBM Corporation, used to support understanding programs written in IBM / 370 assemble language; XREF / XREFDB[2], developed for C++, C and Pascal by Brown University, aiming to support maintaining programs, especially object-oriented programs; OOTM[3], developed by University of Texas at Arlington, mainly supporting object-oriented testing; COBOL/SRE[4], a set of reengineering tools to support reusable component recovery process; and GRASP/Ada[5], focusing on reverse engineering of control structure diagrams from Ada PDL or source code. However, very few of them have been implemented in the context of a tool kit to simultaneously support several research in above software engineering fields, especially those of C++ programs.

Various code analysis tools in different fields usually need some common program information, for example, information on class, inheritance, function and operator overload and message link for C++ programs. Moreover, they even share some same program information to keep presenting consistently and accurately. To support multiple analysis tools based on common program information, it is practical and feasible to store program information into database, thereby avoiding duplicating extraction process for each analysis

---

[1] Sponsored by the Chinese National 9th Five-Year Plan and National 863 Hi-Tech Program.

tool.

We built a tool kit of code analysis for C++ programs, called JBPAS (JadeBird Program Analysis System), which consists of three major components: a C++ front end, an information manager, and a set of program analysis tools. The C++ front end extracts program information from C++ source codes according to a conceptual model and stores it into incremental database. The information manager constructs the program information database through linking incremental databases and provides a database access interface to analysis toolset, which includes various tools used for software testing, software maintenance, reverse engineering, reengineering, and software reuse. All of these tools share the same program information database through the information manager.

This paper describes the program information database and analysis tools based on it. The rest of this paper is organized as follows. First, section 2 presents an overview of JBPAS. Section 3 details JBPAS conceptual model. Section 4 discusses linking incremental databases. Section 5 summaries related work. Finally, section 6 gives the current status and future work.

2. JBPAS Tool Kit

We initially employed the C++ program information database to support research in the area of program understanding, and have implemented the prototype version of a C++ program comprehension system: BDCom-C++[6]. BDCom-C++ consists of three major components: an information extractor, an information manager and a user interface, as shown in figure 1:

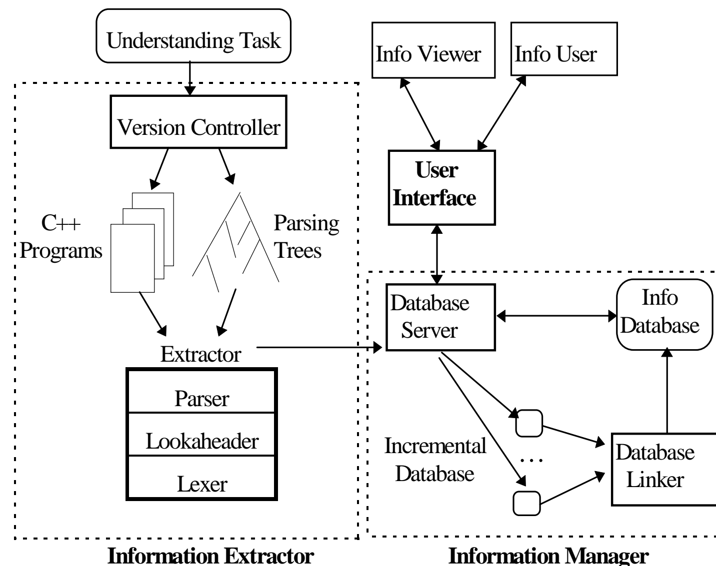

Figure 1. BDCom-C++ System Overview [6]

This system analyzes C++ programs statically in the way of incremental parsing, extracts program information according to a conceptual model formed with Enhanced Entity Relationship[7] model, and stores the information in a relational database. The user interface loads information from the database and presents the program from different perspectives to facilitate program understanding. As shown in figure 1, the center of the BDCom-C++ system is the information manager, which interacts with both the information extractor and the user interface. This approach conforms to the principle of separating the processes of information extraction and presentation, thereby avoiding duplicating extraction process for each analysis tool.

Later we saw several code analysis tools based on the program information database and

anticipate more would appear in the future. Therefore we design a tool kit, called JBPAS, to support research software testing, software maintenance, reverse engineering, reengineering, software reuse, and so on.

Figure 2 shows the architecture of JBPAS:

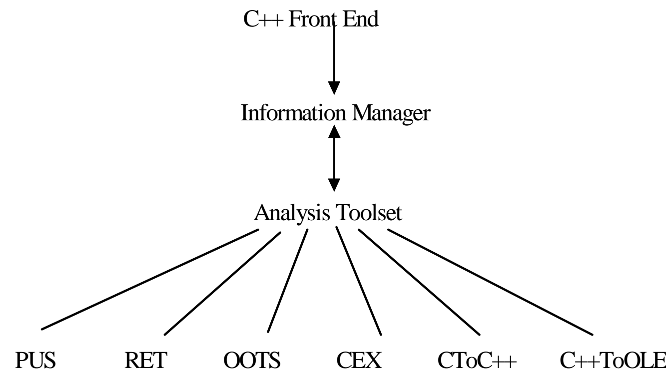

Figure 2. JBPAS Architecture

- PUS: Program Understanding System. PUS facilitates understanding of the function and structure of C++ programs. It loads information from the program database, then organizes and shows the information according to the abstract views needed, and at the same time, switches between the abstract views to represent the program from different perspectives.
- RET: Reverse Engineering Tool. Reverse engineering is the process of analyzing a subject system to identify the system's components and their relationship and create representations of the system in another form or at a higher level of abstraction[8]. RET helps users use the information extracted from the program to recover C++ program's object-oriented design[9] documents, which can be used by forward engineering tools.
- OOTS: Object-Oriented Test Supporter. Traditional testing tools are inadequate for object-oriented programs, because of their new features, such as encapsulation, inheritance, and dynamic binding. OOTS helps to determine test cases based on the program information, and supports C++ program testing, either using white-box or black-box method.
- CEX: Component Extractor. Software reuse, of which an important part is component-based reuse, is considered as a practical and feasible approach to solving the software crisis[10]. Based on program understanding, the component extractor identifies and extracts reusable component through implementing reengineering on the class or class cluster acquired from existing software.
- CToC++: C to C++ Translator. To translate non-object-oriented programs to object-oriented programs is an important way to reuse existing non-object-oriented software. Because C++ is the superset of C, the C++ front end and database server can also apply to C programs. C to C++ translator helps user restructure C programs, locates data structure and its relevant functions, and translates into functionally equivalent C++ program.
- C++ToOLE: C++ to OLE Translator. OLE (Object Linking and Embedding) provides a way to integrate and inter-operate between applications[11]. With the help of the C++ to OLE translator, programmers can encapsulate C++ applications to OLE components conforming to OLE interface standards.

All the above analysis tools share the same program information database through the information manager. And the information manager provides database access interface for other tools, which facilitates the integration of external analysis tools.

3. The JBPAS Conceptual Model

JBPAS forms conceptual model for C++ programs employing Enhanced Entity Relationship (EER) model. According to EER model, C++ programs is viewed as a set of entities and relationships between them, both of which may have a set of attributes. The entities, relationships and attributes of them are just the primary information needed for program analysis in different research.

To support various program analysis tools for different requirements, the conceptual model should be comprehensive and well defined. The EER model of JBPAS is shown in figure 3, in which the rectangle stands for entity and the rhombus means relationship.

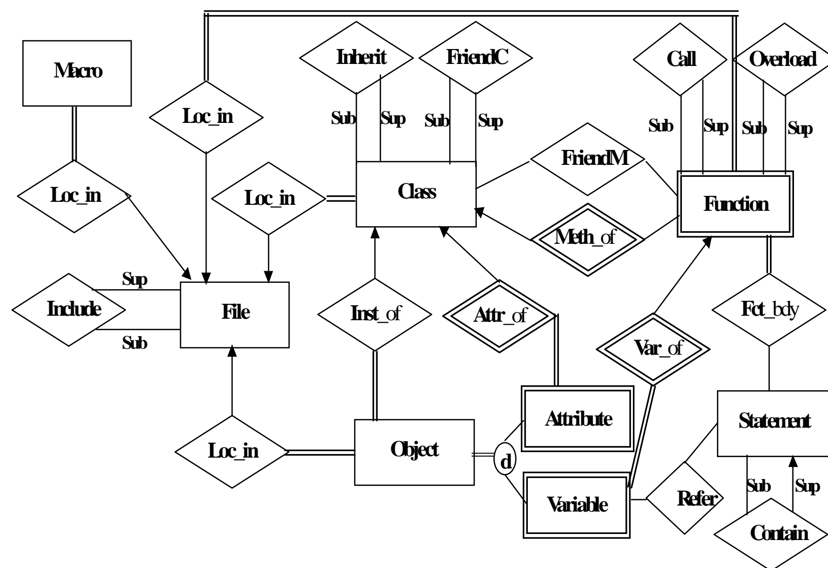

Figure 3. JBPAS Conceptual Model

3.1 Entities

JBPAS 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. Macro name is unique in C++ language, therefore, entity Macro is strong entity (an entity that has a key) and name is its key. So are entity File and entity Class. Entity Statement stands for statements in C++ language and its attribute SID is its key, therefore, entity Statement is also strong entity. On the other hand, in C++ language, functions can have same name due to function overload and they are not unique. Because of scope, attributes and variables can be same and not unique. Therefore, entity Function, entity Attribute and entity Variable have no keys, which means that they are weak entities (an entity that has no key). In EER model, a strong entity is illustrated as a single-line rectangle, while a weak entity is illustrated as a double-line rectangle.

3.2 Relationships

In the JBPAS conceptual model, relationships exist between entities according to their lexical and semantic relation. 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. Similarly, there are semantic relations between classes: inheritance and friend class (We currently ignore nested relation between classes). Because C++ support multi-inheritance, the coordination of relationship Inherit is many-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 Attributes

Each entity has a set of attributes to describe it more completely, and each relationship may also have some supportive attributes. In EER model an attribute is illustrated as an ellipse. Following are attributes of entity Class and relationship Inherit respectively:

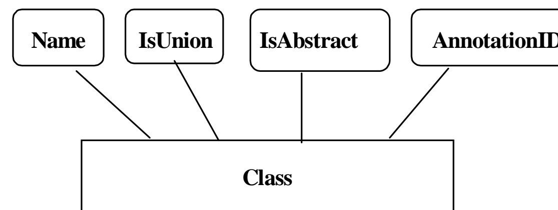

Figure 4. Attributes of Entity Class

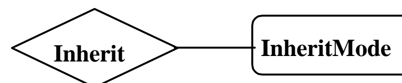

Figure 5. Attributes of Relationship Inherit

4. Incremental Parsing

Since a C++ project may consist of many source files, and when changes are made to some depending source files, it is necessary to reparse those changed files. To avoid reparsing all files, incremental parsing is a practical and feasible approach, which makes it possible to parse only the modified portion. A typical compiler usually creates a .OBJ file for each compiling unit, i.e. a .CPP file in C++ language, while parsing. And then links all .OBJ files to create an executable program or a dynamic-link library. Similarly, JBPAS creates a .IDB file, an incremental database, for each .CPP file in target project to store information on that .CPP file and its included files, and then links all incremental database to construct the large program information database. While reparsing, the version controller locates all files making up of the target project, checks the timestamp of each file, and forwards only those files, which have been modified since the creation of the latest incremental database, to parse. Finally, the database linker relinks all incremental databases to construct the information database.

Apparently, the database linker works in the same way as a compiler's linker, which resolves all external references. However, a code analyzer and a compiler employ different

strategies to generate and use different program information, especially information on declarations. It is common that several .CPP files include same .H files and therefore share same declarations. For example, file COURSE.H has class CCourse's declaration, and is included by file TEACHER.CPP and file STUDENT.CPP, both of which define an object of class CCourse, as shown in figure 6.

```
// COURSE.H
class CCourse{
    char    m_sName[64];
    int     m_nCredit;
    …
};
…
```

File *COURSE.H*

```
// TEACHER.CPP
#include 'COURSE.H'
CCourse teaching;
…
```

File *TEACHER.CPP*

```
// STUDENT.CPP
#include 'COURSE.H'
CCourse studying;
…
```

File *STUDENT.CPP*

Figure 6. Example of Shared Declaration

While compiling each .CPP file, a compiler analyzes declarations and stores their information in the symbol table. The information is used during compiling only the current .CPP file and needn't to store into .OBJ file. After the compiling session, what are unresolved in .OBJ files are external references, and it is just the work of the compiler's linker. However, for each .CPP file, a code analyzer extracts declarations' information and stores it into database permanently for later program analysis. As to the above example, an analyzer stores information on declaration and reference of class CCourse into TEACHER.IDB, the incremental databases of file TEACHER.CPP, and STUDENT.IDB, the incremental databases of file STUDENT.CPP. The two incremental databases may have information similar to figure 7:

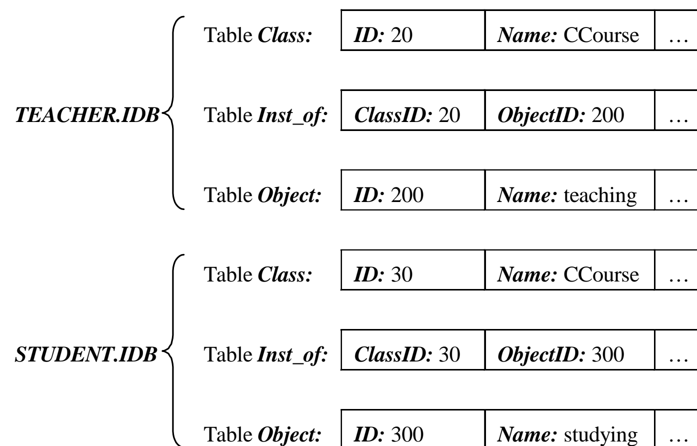

Figure 7. Incremental Databases TEACHER.IDB and STUDENT.IDB

After linking session, the two incremental databases TEACHER.IDB and STUDENT.IDB are linked into one information database. To keep the final information database succinct, the program information database should keep only one copy of information on shared declaration

(We currently ignore the case in which same declarations included by various files are processed differently because of conditional compiling.). Because shared declarations may be refereed by others, the database linker must also update corresponding reference to the shared declaration. For example, after linking TEACHER.IDB and STUDENT.IDB together, the program information may have something like figure 8:

Program Information Database

Table ***Class:***

ID: 20	***Name:*** CCourse	…

Table ***Inst_of:***

ClassID: 20	***ObjectID:*** 200	…
ClassID: 20	***ObjectID:*** 300	…

Table ***Object:***

ID: 200	***Name:*** teaching	…
ID: 300	***Name:*** studying	…

Figure 8. Program Information Database after Linking

5. Related Work

The Ada Maintenance Tool kit (AMT)[12], a set of software tools for analyzing the effect of changes to Ada code, is designed around incremental paring based on structure tree, not on database. Other systems that don't employ database are two cross-reference tools: CSope[13] for C programs and MasterScope[14] for Interlisp programs. The shortcoming of these systems lies in the fact that program information is kept in a specific form difficult to be handled by other tools.

The C and C++ Information Abstractors (CIA[15] and CIA++[16]) are two stand-alone systems that use a database to store extracted program information. However, CIA and CIA++ only form a conceptual model that focuses on global objects (files, macros, global variables, types and functions). The program information is enough to provide extensive coverage of program dependencies, but isn't comprehensive to other applications, say, reengineering. Similar stand-alone systems include FAST[17] for Fortran and OMEGA[18] for Model, a Pascal-like language.

XREF/XREFDB[2] is an integrated system for maintaining object-oriented programs, and is built around a program database. Other systems that are built around a database include ENCORE[19] and the Harvard Programming Development System[20]. However, these systems are designed to a specific requirement and cannot be easily employed widely in other fields.

6. Current Status and Future Work

We have implemented the C++ front end, the information manager, and the prototype versions of three analysis tools: the program understanding system, the reverse engineering tool, and the component extractor. The front end, the information manager and the program understanding system are built through reengineering of the BDCom-C++ system with the help of BDCom-C++ itself. These implemented analysis tools run on PC under Windows 95 platform, and are currently being widely used at CASE Lab, Peking University to analyzing

C++ programs and restructure existing software. The main problem of the system is that for large C++ programs, the program information database becomes considerably huge and suffers from response time.

Our future plans includes the following:

1. Summary the essential information needed by all analysis tools to form a conciser EER model.
2. Enhance the information manager and add a series of query optimizations to increase database performance.
3. Implement other analysis tools of JBPAS cited in section 2, encapsulate all JBPAS tools to OLE applications, and integrate them in an integrated environment.
4. Reuse techniques and code from this system to construct similar systems for other object-oriented language, for example, Smalltalk and Java.

References

[1] L. Cleveland, “A program understanding support environment,” IBM System J., 1989: 28(2): 324-344.

[2] M. Lejter, S. Meyers, and S. P. Reiss, “Support for Maintaining Object-Oriented Programs,” IEEE Trans. Software Eng., 1992: 18(12): 1045-1052.

[3] David Kung, Jerry Gao, Pei Hsia, Yasufumi Toyoshima, Chris Chen, Young-Si Kim, and Young-Kee Song, “Developing an Object-Oriented Software Testing and Maintenance Environment,” Communications of the ACM, Oct. 1995, Vol. 38, No. 10, 75-87.

[4] Jim Q. Ning, Andre Engberts, and Wojtek Kozaczynski, “Recovering Reusable Components from Legacy Systems by Program Segmentation,” Proceedings: Working Conference on reverse engineering, Baltimore, Maryland, May 21-23, 1993, 64-72.

[5] James H. Cross II, “Reverse Engineering Control Structure Diagrams,” Proceedings: Working Conference on reverse engineering, Baltimore, Maryland, May 21-23, 1993, 107-116.

[6] Mei Hong, Yuan Wanghong, Wu Qiong, and Yang Fuqing, “BDCom-C++： A C++ Program Understanding System,” Chinese Journal of Electronics, Vol.6, No.2, April 1997, 64-69.

[7] S. W. Dietrich and F. W. Calliss, “A Conceptual Design for a Code Analysis Knowledge Base,” Software Maintenance: Research and Practice, vol. 4, pp. 19-36, 1992.

[8] E. Chikofsky and J. Cross, “Reverse Engineering and Design Recovery: A Taxonomy,” IEEE Software, 7(1), Jan. 1990, 13-17.

[9] Peter Coad and Edward Yourdon, Object-Oriented Design, Yourdon Press, 1991.

[10] Hafedh Mili, Fatma Mili, and Ali Mili, “Reusing Software: Issues and Research Directions,” IEEE trans. On SE, Vol. 21, No. 6, June 1995, 528-562.

[11] Kraig Brockschmidt, Inside OLE 2, Microsoft Press, 1994.

[12] A. von Mayrhauser, K. Archie , and N. Weber , “Incremental Parsing for Software Maintenance Tools,” J. Systems Software 1993; 23 ;235-243.

[13] J. L. Steffen, “Interactive Examination of a C program with CSope,” In Proc. USENIX Assoc. Winter Conf., Jan 1985, 170-175.

[14] W. Teitelman and L. Masinter, “The Interlisp programming environment,” Computer, Vol. 14, No. 4, Apr. 1981, 25-34.

[15] Y. F. Chen, M. Y. Nishimoto, and C. V. Ramamoorthy, “The C information abstraction system,” IEEE Trans. Software Eng., 1990: 16(5): 325-334.

[16] J. E. Grass and Y. F. Chen, “The C++ information abstractor,” in USENIX C++ Conf. Proc., pp. 265-277, 1990.

[17] J. C. Browne and David B. Johnson, “FAST: A Second Generation Program Analysis System,” In Proc. Second Int. Conf. Software Engineering, 1977, 142-148.

[18] M. A. Linton, “Implementing Relational Views of Programs,” In Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symp. Practical Software Development Environment, May 1984, 132-140.

[19] S. B. Zdonik and P. Wegner, “A Database Approach to Languages, Libraries, and Environments,” Tech. Rep. CS-85-10, Brown University Department of Computer Science, 1985.

[20] T. E. Cheatham, Jr., “An Overview of the Harvard Program Development System,” in Software Engineering Environments, H. Hunke, Ed. New York, North-Holland, 1981.