A Metamodel to carry out Reverse Engineering of C++ Code into UML Sequence Diagrams

Máximo López S., Armando Alfonzo G., Joaquín Pérez O., Juan Gabriel González S., Azucena Montes R.
Researcher of National Center of Research and Technological Development, Computer Science Department,
69042 Cuernavaca, Morelos, México
{maximo, jperez,gabriel,amr@cenidet.edu.mx}
http://www.cenidet.edu.mx

Abstract

When the documentation about the sequence of interactions among the objects in a program is not available but it is required, some work has to be done to document and to obtain the sequence diagrams either by making it in a manual way, or to apply a reverse engineering process that helps to recover the individual diagrams. The present work proposes a computerized method that interprets the characteristics of programs written in C++ code and applies a conversion algorithm that obtains the UML sequence diagrams. To test the method and their conversion algorithm a study case using the work carried out at Gordon College has been taken, which is an automatic teller machine simulator system written in C++.

1. Introduction

Carrying out the maintenance of a software system can be done in two ways: reviewing the designs used in the elaboration of the system or through analyzing the source code. In the first case, it is very probable that the documentation supporting the system's development does not exist and in the second case, reviewing the source code of the system is very complicated, because it requires more time in understanding the way in which it works and to find the specific piece of code that must be modified or replaced.

When there are user's new requirements for the software, this should be revised, adjusted, modified or corrected in its content, so that it fulfills the new proposed expectations.

In this paper we propose a solution to the lack of documentation or information, specifying a metamodel where the characteristics such as the entities, the attributes and the relationships may be incorporated, and to convert this information into UML (Unified Modeling Language) sequence diagrams where interactions among objects can be observed and placed in time sequences; particularly, the diagram shows the objects that participate in the interaction and the message sequences exchanged between them.

2. The Reverse Engineering Processes

Reverse engineering is used in software maintenance phase; since this phase is the biggest and most expensive. After the software first version delivery, the maintenance is extremely much bigger than the phase of initial development.

The process of reconstructing a program design is called "design recovery" and it is an element of the reverse engineering process. The design recovery recreates design abstractions from a combination of: code, existing design documentation (if available), personal experience and general knowledge of the problem and the application domain [2].

The reverse engineering model proposed by Byrne [3] consists of extracting information of the detailed design and its high level abstraction from source code and existing design documents. This information includes a structure representation, data description and a detailed description of the processes. The proposed activities for the design recovery of a software system in the Biggerstaff model [2] are oriented to programs written in a structured language like the C language.

Proceedings of the Electronics, Robotics and Automotive Mechanics Conference (CERMA'06)
0-7695-2569-5/06 $20.00 © 2006 IEEE

3. From Static Analysis to Sequence Diagram

The solution strategy for the reverse engineering of the sequence diagrams consists of a source code modeling, that is, a metamodel that represents an object-oriented program. The translation to sequence diagrams is based on the source code syntax, that extracts such information as the names of the classes, objects, functions and calls, which then are stored in the metamodel as a representation of the code. The viewer developed in [4] is used as a model of the sequence diagrams, from it, the information or elements that are required to build a diagram are obtained. Once the information is recovered and placed in the metamodel, it is transformed into a sequence diagram by means of conversion algorithms. These algorithms have the capacity to reduce the complexity of the obtained diagrams, since they are able to select the method, the classes and to analyze the depth levels of the calls. The conceptual approach for the solution method is shown in Fig. 1. In this work, the input source code files are written in the programming language C++. First, the syntactic analyzer extracts and interprets the information of the source code and then stores it in the metamodel to represent the object-oriented program, after this, an algorithm or conversion program is applied to the obtained representation, to build, visualize and reduce the sequence diagrams corresponding to the input program.

The translator implementation including the metamodel and the conversion algorithms were developed in C++. The syntactic analyzer was made with the help of a parser generator called Antlr [5] using the C++ grammar. In the construction and visualization of the sequence diagrams the MFC (Microsoft Foundations Classes) was used.

3.1 The Sequence Diagrams Metamodel

A sequence diagram is an interaction diagram that exhibits the temporary order of the messages. A sequence diagram presents a set of objects and the messages sent and received by them. The objects are usually instances of classes (with names or without names), but they can also represent instances of other elements such as: collaborations, components and nodes [6].

The metamodel developed by Macias Alonso [4] is readjusted in this work to visualize the sequence diagrams.

To build a sequence diagram the following information is required: name of the classes that intervene in an interaction, names of the messages that are carried out -such as regular calls or the calls to functions of the same class, and all the messages in the proper order.

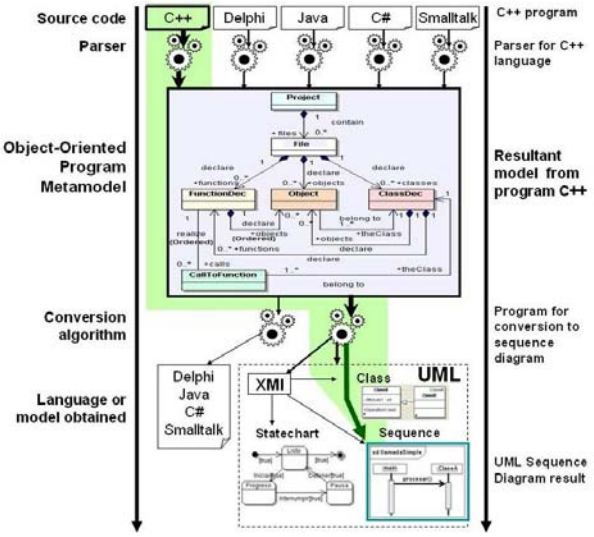


Fig. 1. Conceptual model of the solution method

3.2 The Translator and the Metamodel for an Object-Oriented Program

During the syntactic analysis the translator extracts, resolves references and interprets information in the code useful for the sequence diagram; also, the sequence diagram requires information about classes and messages.

The translator resolves references to objects, as it is the case of calls where it is required to know to what class they belong, simply by just identifying the object that precedes the object and in more difficult cases, it is solved at the end of the analysis.

When the analysis of the program has concluded an instance of the class diagram has been created (Fig. 2).

The metamodel instance for the previous diagram is shown in the Table 1. In the left side the metamodel instance is shown and in the right side its source code written in C++.

Once the syntactic analysis of the program is concluded, the translator keeps in a file the information about all the classes, functions, objects and calls that were extracted in the process, so that later on, the tool viewer can process and publish the information.

4. Conversion Algorithms

Once the information is recovered from the C++ code and deposited in the metamodel as a program structure, it is necessary to convert it into the corresponding sequence diagram proposed by Macias Alonso [4]. To do this, an algorithm or program which

enables to visualize the diagram is used. Such a program is able to look for interactions carried out in a method and the classes that intervene in the interactions, to distinguish calls to functions of the same class and even, to introduce a generic class for those methods that do not belong to a specific class as it is the case of the *main* code. Lastly, the algorithm is able to reduce the diagrams establishing a level of depth in the calls.

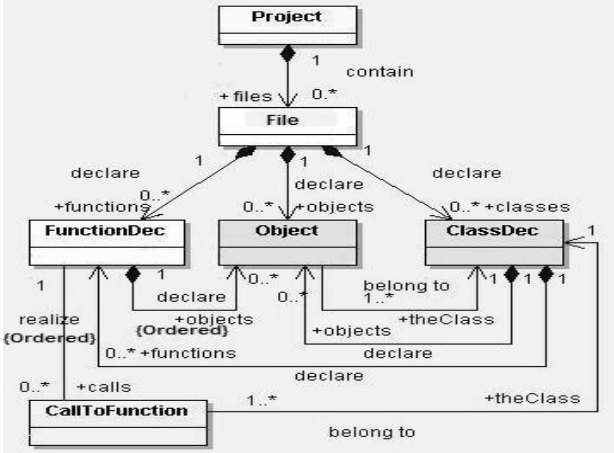


Fig. 2. Object-Oriented Program Metamodel

5. Case Study

An Automated Teller Machine (ATM) simulator system has been selected as a source of study cases, considering only the software part of the system. This work was developed at the Gordon School in the Mathematics and Computer Sciences Department (http://www.math-cs.gordon.edu/courses/cs211/ATMExample/index.html) [7] by Professor Russell C. Bjork for his object-oriented software development course. This system provides the following functions: System Startup, System Shutdown, Session, Transaction, Withdrawal, Deposit, Transfer and Inquiry.

A full version of the system was developed in Java and a partial version is available in standard C++ (for UNIX systems).

5.1 Study Case: Invalid PIN

An invalid PIN extension [7] is started from within a transaction when the bank reports that the customer's transaction is disapproved due to an invalid PIN. The customer is required to re-enter the PIN and the original request is sent to the bank again. If the bank now approves the transaction, or disapproves it for some other reason, the original use case is continued; otherwise the process of re-entering the PIN is repeated. Once the PIN is successfully re-entered, it is used for both the current transaction and all subsequent transactions in the session. If the customer fails three times to enter the correct PIN, the card is permanently retained, a screen is displayed informing the customer of this and suggesting he/she contact the bank, and the entire customer session is aborted. If the customer presses Cancel instead of re-entering a PIN, the original transaction is cancelled.

A sequence diagram obtained by the inverse engineering process of the present work is shown in Fig. 3.

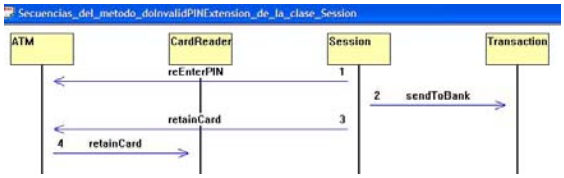


Fig. 3. Reverse engineering when invalid PIN

5.2 Study case: Start Session

A session [7] is started when a customer inserts an ATM card into the card reader slot of the machine. The ATM pulls the card into the machine and reads it. (If the reader cannot read the card due to improper insertion or a damaged stripe, the card is ejected, an error screen is displayed, and the session is aborted.) The customer is asked to enter his/her PIN, and is then allowed to perform one or more transactions, choosing from a menu of possible types of transaction in each case. After each transaction, the customer is asked whether he/she would like to perform another. When the customer is through performing transactions, the card is ejected from the machine and the session ends. If a transaction is aborted due to too many invalid PIN entries, the session is also aborted, with the card being retained in the machine.

6. Related works

The commercial tools [12, 13, 14, and 15] and the related works [8, 9, 10, and 11] have several related characteristics but few strategies to carry out reverse engineering of UML sequences diagrams. The differences are summarized and compared according to the following criteria:

1. Recovery strategy: For the dynamic analysis of a complete system, the information recovery from the code is carried out by means of code instrumentation and static analysis for full or partial systems using compilers techniques.

2. Visualization of the flow of control: It is possible to express the flow of control of a method by means of the sequence diagrams; there are few studies for the UML 2.x version and some cases are proposed in [15] for the UML version 1.x.
3. Diagrams format: The tool's own format is used or an image format, just as JPG; few people use the data exchange format (XMI 1.x standard) and almost none use XMI 2.x.
4. Programming language: the tools analyze more code written in Java language and less for C++ and C #.
5. Reduction strategies: In the diagrams reduction, few tools use filters for classes and nesting levels in the calls for a method in particular. In the commercial tool of [14], the messages that repeat have a special reduction.
6. Versions of sequence diagrams: in their majority for the UML version 1.x.
7. Translation strategy: The use of metamodels in the code translation to sequences diagrams is not popular. Only one work [9] uses an execution lines metamodel to carry out the dynamic analysis and the rest of the works don't use metamodels for the translation or they simply don't mention it.

The sequence diagram obtained for the study case Start Session is shown in Fig. 4.

7. Conclusions

The reverse engineering is used to understand the structure and behavior of software whose documentation is not currently updated. This paper is a starting point for other research works on reverse engineering for code written in C++ in which a metamodel is used. Thus it is possible to observe that from code written in C++ language the UML version 1.x sequence diagrams can be recovered. The developed tool recovers relevant information from the code in the form of sequence diagrams.

This implies that to carry out reverse engineering from source code written in C++ language and to obtain the sequence diagrams of a system, the following has to be fulfilled: (1) To work with a full system or with a partial one, either currently in operation or in file, (2) To obtain the diagrams according to the users needs, where it is allowed to select to the classes and methods that interest to them and help to reduce the diagrams complexity.

One of the fundamental problems was the information recovering and how to visualize it. For the first case, we created a structure of classes that approaches us to the meaning of the code; this was solved with the metamodel which is focused on the Object-Oriented characteristics of a program. In the metamodel we store the relevant information that the sequence diagrams need or require, for example: the calls made within a method become messages within the sequence diagrams.

In our approach, once we have an instance of the code, we use a conversion algorithm that generates a sequence diagram. The algorithm is able to look for interactions made in a method and the classes that take part in the interactions, to identify self o recursive calls within a class, and to reduce to the diagrams by establishing a depth level for the calls. The algorithm introduces a generic class for those methods that do not belong to a specific class as it is the case of *main*, which makes use of objects in the program.

We have carried out reverse engineering to several study cases in a gradual way, from some very simple interaction cases to others very complex.

It follows that the main contribution presented in this paper is the concept of a code metamodel for OO programs, in specific, for those C++ language codes; and the conversion algorithms to get sequence diagrams.

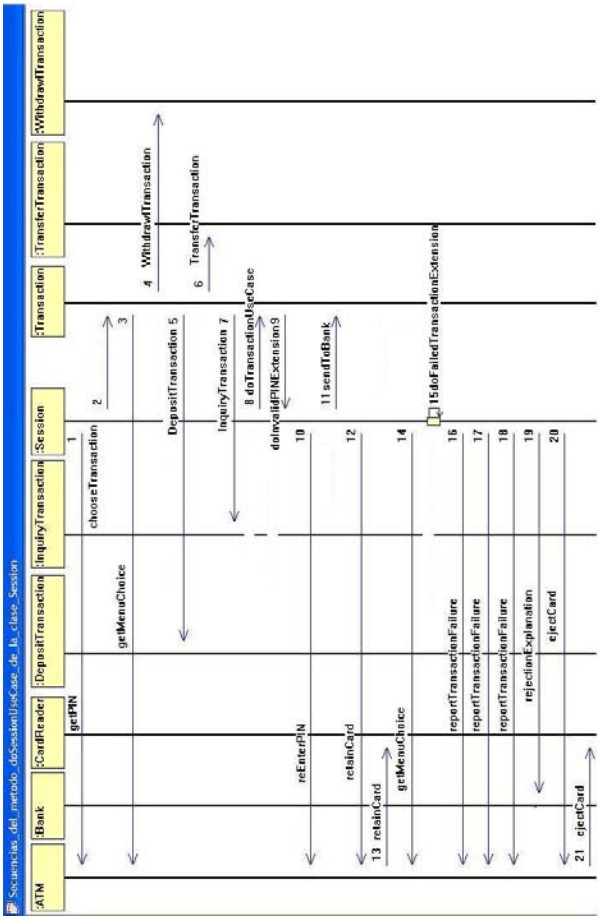


Fig. 4. Reverse engineering for Start Session

References

[1] Tonella, P., and Potrich, A., *Reverse Engineering of Object Oriented*, Springer New York , 2005

[2] Biggerstaff, T.J., *Design Recovery for Maintenance and Reuse*. IEEE Computer , 1989

[3] Byrne, Eric J., *Software Reverse Engineering: A Case Study*. Software-Practice and Experience, Vol. 21, 1991

[4] Macias J. Alonso, *Ambiente de modelado y diseño de sistemas de software utilizando diagramas de secuencias*, Tesis de maestría en ciencias, Centro Nacional de Investigación y Desarrollo Tecnológico, México, August 2004.

[5] *ANother Tool for Language Recognition*. http://www.antlr.org/grammar/cpp Abril 2005

[6] Hans-Erik Eriksson, Magnus Penker. *UML Toolkit*. Wiley Computer Publishing, 1998.

[7] *ATM Example System* http://www.math-cs.gordon.edu/courses/cs211/ATMExample/index.html

[8] Raitalaakso Timo, *Dynamic Visualization of C++ Programs with UML Sequence Diagrams*, Tesis de maestría en ciencias, Universidad de Tecnologica de Tampere, Finlandia, 2000. http://practise.cs.tut.fi/pub/ August 2004

[9] L.C. Briand, Labiche, and Miao, "Towards the Reverse Engineering of UML Sequence Diagrams", *Proc. IEEE International Conference on Reverse Engineering*, Victoria, BC, Canada, 2003.

[10] Tonella P., and Potrich A., "Static and Dynamic C++ Code Analysis for the Recovery of the Object Diagram", *International Conference on Software Maintenance*. Canada, 2002. http://www.cis.ohio-state.edu/~rountev/788/private/papers/tonella_icsm02.pdf March 2004.

[11] Volgin Olga *Analysis of flow of control for reverse engineering of sequence diagrams*, Tesis de maestría, Universidad del Estado de Ohio, Estados Unidos, April 2004. http://www.cse.ohio-state.edu/~rountev/publications.html Noviembre 2004.

[12] *Logsequencer*. http://www.logsequencer.com/ April 2004.

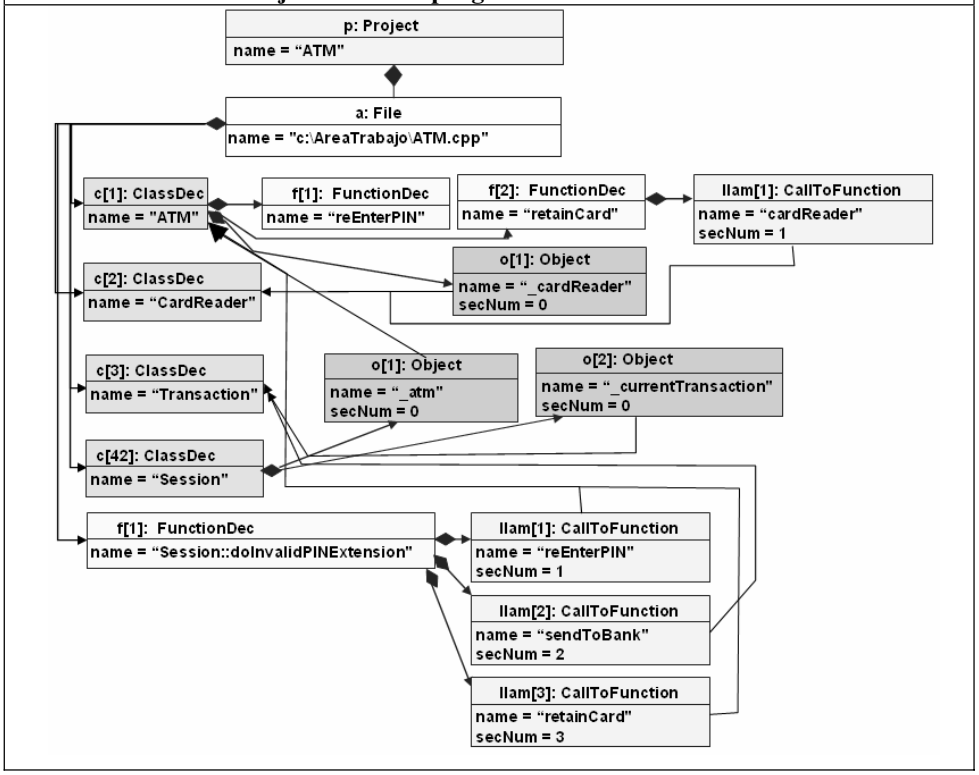
[13] *Nasra Consulting,* http://www.nasra.fr/ May 2004.

[14] Logic Explorers Inc., http://www.logicexplorers.com/ March 2004.

[15] Borland Inc., http://www.borland.com/together/ September 2004

Table 1. Metamodel instance of an object-oriented program written in C++

Object-oriented program metamodel instance



C++ Source Code

```
class CardReader { /* PURPOSE: Model the ATM component that reads a customer's card */
public:
  CardReader();
  void ejectCard();                    void retainCard();
  enum ReaderStatus { NO_CARD, UNREADABLE_CARD, CARD_HAS_BEEN_READ };
  ReaderStatus checkForCardInserted();
  int cardNumber() const;
private:
  ReaderStatus    _status;
  int             _cardNumberRead;       };
class ATM {  /* PURPOSE: Manage the overall ATM and its component parts */
public:  // Interact with the customer in support of various use cases
  int getPIN() const;
  int ATM::reEnterPIN() const {
    //_display.requestReEnterPIN();
    //int PIN = _keyboard.readPIN(_display);
    //_display.clearDisplay();
    return 0;        };
  void ejectCard() const;
  void retainCard() const {
    _cardReader.retainCard();          };
  int number() const { return _number; };
private:
  int                _number;
  const char *       _location;
  CardReader &       _cardReader;   };
class Session;
class Transaction {  /* PURPOSE: Serve as base class for specific types of transactions */
public:
  Status::Code doTransactionUseCase();
  virtual Status::Code getTransactionSpecificsFromCustomer() = 0;
  virtual Status::Code sendToBank() = 0;
  virtual Status::Code finishApprovedTransaction() = 0;
protected:
  Session &       _session;
  ATM &           _atm;
  Bank &          _bank;
  int             _serialNumber;
private:
  static int      _lastSerialNumberAssigned;     };
class Session {  /* PURPOSE: Manage a session with a single user. */
public:
  Status::Code doInvalidPINExtension();
  int cardNumber() const { return _cardNumber; };
  int PIN() const { return _PIN; };
  enum { RUNNING, FINISHED, ABORTED }  _state;
private:
  int          _cardNumber;
  ATM &        _atm;
  int          _PIN;
  Transaction *_currentTransaction;              };
Status::Code Session::doInvalidPINExtension() {
  Status::Code code;
  for (int i = 0; i < 3; i ++) {      PIN = _atm.reEnterPIN();
                                      code = _currentTransaction -> sendToBank();
                                      if (code != Status::INVALID_PIN)  return code;         }
  _atm.retainCard();            _state = ABORTED;            return Status::INVALID_PIN;    }
```