Multi-Paradigm Spreadsheet for End Users

Jong-Myung Choi,[†] *and Young-Chul Kim*[††]*,*

Department of Computer Engineering, Mokpo University, South Korea[†]
Department of Electronic Commerce, Yuhan College, South Korea[††]

Summary
Spreadsheets are widely used in various areas by end users because they are easy to use. However, due to the lack of the methodology for analysis and design for spreadsheet applications, users suffer with problems such as high error rates, low reusability, and maintenance difficulties. To mitigate these problems, we introduce a multi-paradigm spreadsheet that allows end users to analyze their problems and design for the solutions. Our multi-paradigm spreadsheet supports object-oriented programming, flowchart, XML, and spreadsheet. While writing the XML document, they analyze the problems by grouping similar things into one concept, and the concept develops into a class in object-oriented programming. The flowchart and spreadsheet are used for describing operations or methods of the class. Our multi-paradigm spreadsheet is so simple that even end users can analyze and design for their applications.

Key words:
Multi-Paradigm, Spreadsheet, End Users, OOP

1. Introduction

The Spreadsheet is one of the most popular computer softwares because of its user friendliness, and it allows even end users to write applications for auditing, calculation, or statistics. However, the applications are developed without any systematic analysis or design [1] because most of their developers are non-programmers. As the number of end-user developers increases, the need for a methodology that even end users can follow easily is increasing [2].

Spreadsheet applications developed in ad-hoc manner have serious problems. First of all, these applications suffer with high error rates. In fact, according to Panko's researches [3,4] 20% to 40% of all spreadsheet applications contain errors. Furthermore, these errors cause serious financial loss. For example, they found 131 errors in the spreadsheet application used for tax calculation in UK, and there was a £196,603 loss per error [5].

Second, a spreadsheet provides only limited support for reuse. Other programming languages allow users to build libraries and reuse them. In particular, object-oriented languages support not only libraries but also inheritance and composition mechanisms for reuse. However, spreadsheet supports only copy-paste and macros mechanism for reuse. Furthermore they are even apt to cause errors.

Finally, a spreadsheet provides limited mechanisms for modularity and abstraction. Because it does not provide high-level abstraction, users have difficulties in understanding or solving problems. In order to figure out the overall structure of the application, the users must repeatedly select a cell, read the formula, and move on to the next cell, until they have seen enough formulas to get an overview of the spreadsheet [6].

These problems prevent spreadsheet from being used in the development of large applications. In order to solve these problems, we introduce an object-oriented spreadsheet system. It allows users to think about the application problems in high conceptual level abstraction, to analyze and design applications according to software engineering principles, and to reuse software modules. Markku's research [7] shows that the different paradigms in the spreadsheet system affect the error rates, and that the high conceptual level reduces error rates in the spreadsheet. Object-oriented programming in spreadsheets will reduce errors, increase reusability, and help users to write high quality programs.

The rest of this paper is divided into 4 sections. In the next section, we show how to model a spreadsheet program as class using XML, and then we introduce object-oriented programming in a spreadsheet. We then discuss related studies. In the last section, we draw some preliminary conclusions.

2. Data Modeling

There has been some research on methodologies for spreadsheet modeling [1,8]. Most of them are basically based on the structured analysis and design methodology [8], but nowadays object-oriented programming is widely used, and it is much easier for end users because it supports natural modeling, reusability, and maintainability. Therefore, we set our goal of achieving user-friendly analysis, design, and programming by combining the merits of object-oriented programming and spreadsheet programming.

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

Since most spreadsheet users are non-programmers, and they do not have knowledge of object-oriented programming, a simple method is needed that helps users in identifying and constructing classes without expert knowledge. In order to meet this requirement, we decide to use XML [9] as a means of structuring data and defining classes. Because XML to object mapping, such as JAXB [10], is commonly used, it is reasonable. Furthermore, since the objects used in a spreadsheet are typically entity objects, the mapping is rather natural. However, the XML application should be simple enough for end users to use.

Let me give an example. Assume that we process the student's score and grade with our approach. First, we advise users to write an example XML document for a student. Writing an example document helps users in analyzing the problem, and structuring data according to their local meaning and coherence. While they handle concrete data, they will feel at ease. An XML document for a student can be written as:

```
<Student>
  <name>Gil D. Hong</name>
  <math>85</math>
  <eng>90</eng>
  <avg>
    avg = (math + eng) / 2
  </avg>
</Student>
```

The example XML document is transformed into our XML application, called TCML (Tiny Class Markup Language). It has minimum rules for end users. In TCML, end users can choose their own tag names and write documents with those tags as long as you follow some rules on predefined attributes. For example, "type" attribute is predefined, and it means the data type of a tag contents. In the example XML document, the tag contents are converted into the "type" attribute according to its data type. The type attribute's value "abc" means that the data type is a string. Similarly, "99" and "99.99" means integer and real number relatively. If the "type" attribute is omitted, the data type is considered to be the same as the tag name.

End users can write either an example XML document or a strict TCML document. For example, the student example document is transformed as follows:

```
<Student>
  <name type="abc"/>
  <math type="99"/>
  <eng type="99"/>
  <avg type='99.99" readonly="yes">
    avg = (math + eng) / 2
  </avg>
</Student>
```

The TCML document is converted into java classes. The topmost tag corresponds to class name, and its child tags match the class's member fields. If a user needs to describe a method or operation, he/she can describe the formula in text or the algorithm using the flowchart. The operations in a spreadsheet are rather simple, and they are in the form of $x = f(x_1, x_2,.., x_n)$. It means that most spreadsheet formula can be represented as a function of member fields within a class. Therefore, in TCML, the operations are specified as the contents of a tag. For example, the "avg" tag has the text contents that define how the value should be calculated. Furthermore, multi-paradigm spreadsheet provides a graphic flowchart to help users in describing complex operations.

Inheritance is a very powerful mechanism for reuse in object-oriented programming. A child class inherits member fields and methods from the parent class. In TCML, the inheritance can be denoted with the "kindof" attribute. For example, the Student class inherited from the Human class can be described as:

```
<Human>
   <name type="abc"/>
</Human>
<Student kindof="Human"> ...
</Student>
```

Composition is another mechanism for reuse in object-oriented programming. Composition makes it possible to build a big class composed of other objects. Composition is easily expressed with nested tags in XML documents. For example, the ClassRoom class consists of students and score average. Then, the example document for the ClassRoom can be described as:

```
<ClassRoom>
   <Student>
     <name>Gil D. Hong</name> ...
   </Student>
   <Student>
     <name>Chul S. Kim</name> ...
   </Student>
   <avg>
     avg = sum(Student.avg)/numberof(Student)
   </avg>
</ClassRoom>
```

The example document is transformed into the following intermediary TCML document.

```
<ClassRoom>
  <Student number="many"/>
  <avg type="99.99" readonly="yes">
```

```
   avg=sum(Student.avg)/numberof(Student)
  </avg>
</ClassRoom>
```

The "number" attribute denotes the cardinality. The repeated tags in the example document (eg. Student) are condensed to one tag with the "number" attribute. TCML provides some predefined functions for collective data. The "sum" and "numberof" are examples of the predefined functions. The "sum" function returns summation, and the "numberof" function return the amount of data. The class designer parses a TCML document, analyzes its meaning, and converts it into a strict TCML document. The XML parser checks the validity of the strict TCML document. The strict TCML has the following DTD.

```
<!ELEMENT class   (field*, method*)>
 <!ATTLIST  class
        abstract NMTOKEN  #IMPLIED
        name   NMTOKEN  #REQUIRED
        kindof  NMTOKEN  #IMPLIED>
 <!ELEMENT field   (#PCDATA)>
 <!ATTLIST  field
        name    NMTOKEN #REQUIRED
        type    CDATA    #IMPLIED
        number  NMTOKEN #IMPLIED
        readonly NMTOKEN #IMPLIED>
 <!ELEMENT method  (#PCDATA)>
 <!ATTLIST  method
        name    CDATA   #REQUIRED
        type    CDATA   #REQUIRED
        args    CDATA   #IMPLIED>
```

The strict TCML document is again converted to a Java class. The generated class has setter and getter methods for each member field. However, a tag with a "readonly" attribute has only a getter method. The generated getAvg() method of the Student class is as follows:

```
public double getAvg() {
   avg = (math + eng) / 2.0;
   return avg;
}
```

TCML is very simple and limited because it is designed for novices. Experts can modify the generated class on their own, or they can directly write Java classes. Java classes can be used in the object spreadsheet.

3. Multi-Paradigm Spreadsheet

3.1 Objects in Spreadsheet

In order to apply object-oriented programming to a spreadsheet, we should be able to represent classes and objects in it. In a multi-paradigm spreadsheet, classes are represented in a horizontally contiguous area. Fig. 1 shows how the Student class is presented on the spreadsheet system. The read-only member fields have the 'R' symbol on the upper right corner of spreadsheet cells.

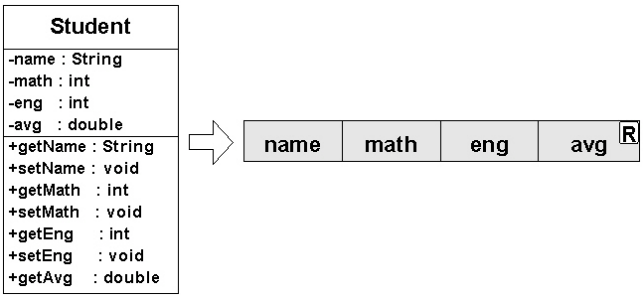

Fig. 1 Class Declaration in Spreadsheet

A class plays the role of model or template for data, and data located in the vertically contiguous area below the class declaration is considered as the instances of the class. Fig. 2 shows the class declaration and its instances in the object spreadsheet. When the data is input, the setter methods are called automatically. On presentation of data, the getter methods are called. For example, when end users key in the student's name, the set method is called. In addition, whenever the Math or English score is changed by user input, the set methods are called, and the average is calculated automatically. This removes the need for users to type the formulas, and this will reduce the mechanical errors in spreadsheets. The "avg" member field does not have a setter, so that users cannot input data into the "avg" cell. This mechanism prevents users from miss-typing in the wrong cell.

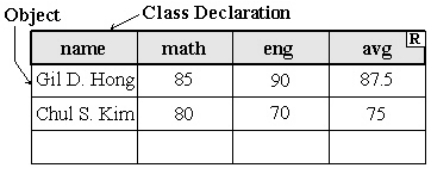

Fig. 2 Class and Objects in Object Spreadsheet

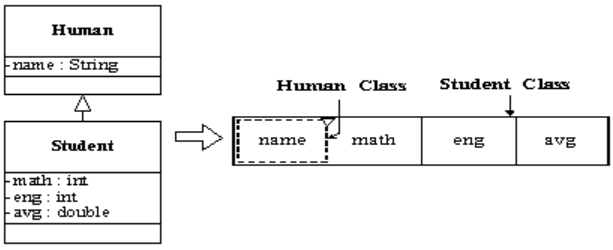

Fig. 3 Inheritance

We can represent inheritance in the spreadsheet as shown in Fig. 3. We use a triangle symbol on the upper right corner of the parent class to identify inheritance. Composition is represented by a diamond symbol. Fig. 4 shows inheritance and composition in the spreadsheet. Using the object spreadsheet, users can describe class declaration, objects, inheritance, and composition in the spreadsheet. It means that they can apply object-oriented technology to developing spreadsheet applications. Programmers are able to use object-oriented tools such as UML. Systematic object-oriented analysis and design will reduce logic and omission errors in spreadsheets.

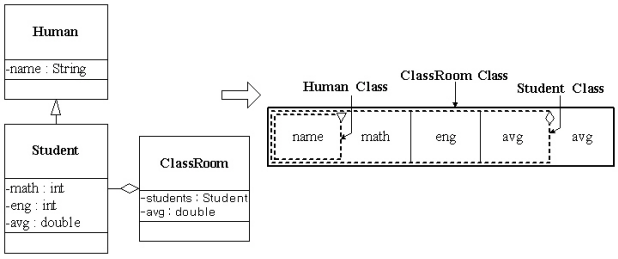

Fig. 4 Composition

3.2 System Architecture

The multi-paradigm spreadsheet system supports both the spreadsheet paradigm and the object-oriented paradigm. To make this possible, we determine to separate logic and presentation in the spreadsheet application. The system is designed and implemented according to the MVC (Model-View-Controller) design pattern. View plays the role of presentation. Model plays the part of logic for the application. Fig. 5 shows the architecture of the multi-paradigm spreadsheet system.

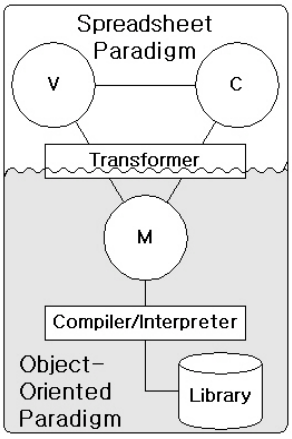

Fig. 5 Architecture of Multi-paradigm Spreadsheet

View provides the GUI of the traditional spreadsheet system. Controller interacts with users, and changes the state of objects. Model maintains data and program logic, and performs operations. The classes in the model are written in the Java programming language, and we choose this language for its portability. The interaction is converted into the object-oriented language code by the transformer, and the Jython interpreter [11] executes the code. Fig. 6 shows the class designer and the flowchart. The class designer converts the XML documents into the Java program. The flowchart is used to describe methods in the class designer.

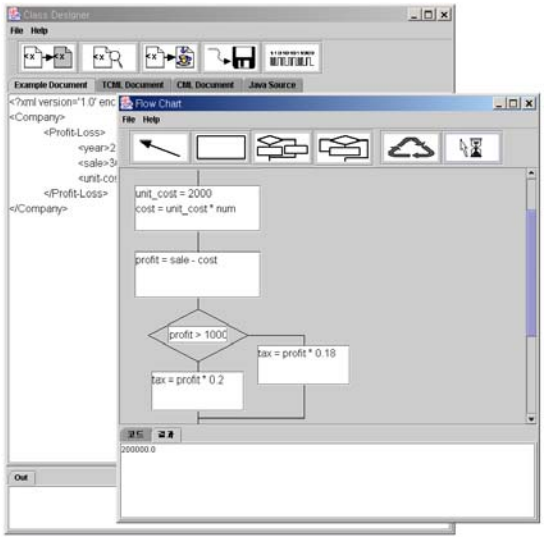

Fig. 6 Class Designer and Flowchart

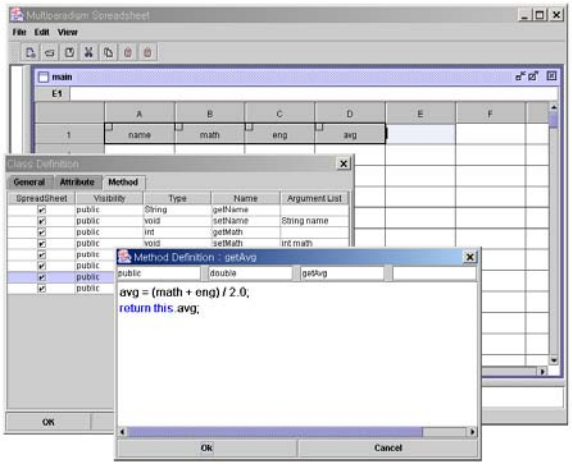

Fig. 7 Multi-paradigm Spreadsheet System

Fig. 7 shows the appearance of the multi-paradigm spreadsheet. When end users key in data into the cell, a

new instance is created at first, and the value of the member field is assigned by calling the setter method. While keying in data, the calculation is performed automatically. The viewing of the member fields on the spreadsheet is done by the calling getter methods.

4. Related Works

Due to the success of the spreadsheet, there has been much research on spreadsheets. Some studies tried to apply the structured analysis and design methodology to the spreadsheet in order to reduce logical errors [12,13]. These studies differ from our study in that we use multi-paradigm programming and the object-oriented methodology for end users.

Some other studies [2,14,15] are concerned with reducing or preventing error by providing auditing or assertion mechanisms. These studies are very useful for current spreadsheet systems, but they also have the limit of not being suitable for large applications.

There also has been research on multi-paradigm programming in spreadsheets. Michael [16] applied Logic Programming to spreadsheets, and Chris [17] and Bjorn [18] combined spreadsheet and Functional Languages. These works are very interesting, but it is rather hard for end users to use these languages.

Other researchers worked on the object-oriented spreadsheet. Model Master [19] and ASP [20] are examples. Model Master is a compiler that generates spreadsheet equations from textual specifications of models. Model Master provides inheritance, in the object-oriented sense of the word, enabling the users to take a partially specified object, and extend it by adding more attributes or equations. ASP [20] is a Smalltalk-based object-oriented spreadsheet. Cells in ASP can literally hold any type of Smalltalk object. The formula syntax of ASP is the Smalltalk language with a single added construction. These object-oriented spreadsheets are similar to our multi-paradigm spreadsheet, but our system supports user-friendliness by using XML, flowchart, and a spreadsheet user interface.

5. Conclusions

Because of the user-friendliness of a spreadsheet, it is widely used by end users. Inherently, they are not concerned with systematic analysis or design for their applications. Due to their ad-hoc programming style, they suffer serious problems. First of all, spreadsheet applications contain many errors, and the errors cause critical financial loss. Second, a spreadsheet hardly supports reuse. It supports only copy-paste, and the low reusability prevents spreadsheets from being used for large applications. Finally, it is very hard to understand the spreadsheet application's structure and logic because of its complex data dependency in formulas.

To reduce these problems, we proposed a multi-paradigm spreadsheet system. It consists of a class designer and an object spreadsheet. The class designer allows end users to define classes using XML and flowchart. The XML documents are transformed into Java classes automatically. When the classes are used in the spreadsheet, their instances are created, and their setter and getter methods are invoked whenever users input data.

We separate logic and presentation for implementing the spreadsheet system. In presentation, the class declaration is presented on the spreadsheet in the horizontally contiguous area, and the instance is located in the vertically contiguous area below the class declaration. The logic part maintains the class definition and class libraries.

The multi-paradigm spreadsheet allows users to use object-oriented analysis, design, programming, and maintenance. In addition it reduces logic and omission errors by supporting object-oriented technology. It also reduces mechanical errors by removing the need for formula input and by preventing users from slips in "readonly" data. When a user defines classes using RMI or CORBA, the multi-paradigm spreadsheet system can be used in distributed computing.

References

[1] Ronen, B., Michael, A. P., Henry, C. L.: Spreadsheet Analysis and Design, *Comm. of ACM*, Vol. 32, No. 1 (1989) 84-93

[2] Margaret Burnett, Curtis Cook, and Gregg Rothermel, "End-user software engineering", *Comm. of ACM*, Vol. 47, No, 9, pp. 53-58, Sep., 2004.

[3] Panko, R. R.: Spreadsheet Research (SSR) Website (http://www.cba.hawaii.edu/panko/ssr/) Honolulu, Hawaii: University of Hawaii

[4] Panko, R. R., Halverson, R. P. Jr., "Spreadsheets on Trial: A Framework for Research on Spreadsheet Risks", *Proc. of the 29th Hawaii International Conference on System Sciences* , 1996.

[5] Raymond J. Butler, "Is This Spreadsheet a Tax Evader? How H.M. Customs & Excise Test Spreadsheet Applications", *Proc. Of the 33rd Hawaii International Conference on System Sciences*, 2000.

[6] Igarashi, T., et al., "Fluid Visualization of Spreadsheet Structures", *Proc. 14th IEEE Symposium on Visual Languages*, pp. 118-125, 1998.

[7] Tukianen, M.: Uncovering Effects of Programming Paradigms: Errors in Two Spreadsheet Systems, *Proc. of PPIG-12 12th Annual Meeting of the Psychology of Programming Interest Group*, pp. 247-265, 2000.

[8] Knight, B., Chadwick, D., Rajalingham, K., "A Structured Methodology For Spreadsheet Modeling", *Proc. of the EuSpRIG 2000 Symposium on Spreadsheet Risks, Audit and Development Methods*, pp. 43-50, 2000.

[9] XML, available at http://www.w3.org/xml/
[10] Sun, *Java Architecture for XML Binding*, available at http://java.sun.com/xml/jaxb/.
[11] Hugunin, J., "Python and Java: The Best of Both Worlds", *Proc. of the 6th International Python Conference*, 1997, available at http://jpython.org/.
[12] Kamalasen Rajalingham et al, "Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development", *Proc. of the 33rd Hawaii international Conference on System Sciences*, 2000.
[13] Brian Knight, David Chadwick and Kamalasen Rajalingham, "A Structured Methodology for Spreadsheet Modelling", in *Proc. of EuSpRIG*, pp. 43-50, 2001.
[14] Takeo Igarashi et al, "Fluid Visualization of Spreadsheet Structures", *Proc. of IEEE Symposium on Visual Languages*, pp. 118-125, 1998.
[15] Gregg Rothermel, et al, "A Methodology for Testing Spreadsheets", *ACM Transactions on Software Engineering and Methodology*, Vol. 10, No. 1, pp. 110-147, 2001.
[16] Michael Spenke and Christian Beilken, "A Spreadsheet Interface for Logic Programming", *Proc. of CHI*, pp. 75-80, 1988.
[17] Chris Clack and Lee Braine, "Object-oriented functional spreadsheets", *Proc. of the Glasgow Workshop on Functional Programming*, pp. 1-12, 1997.
[18] Björn Lisper and Johan Malmström, "Haxcel: A Spreadsheet Interface to Haskell", *Proc. of International Workshop on the Implementation of Functional Languages*, pp. 206-222, 2002, available at http://www.mrtc.mdh.se/.
[19] Paine, J., "Model Master: an object-oriented spreadsheet front end", *Proc. of CALECO97*, 1997.
[20] Piersol, K. W., "Object Oriented Spreadsheets: The Analytic Spreadsheet Package", *Proc. of OOPSLA*, pp. 385-390, 1986.

Jong-Myung Choi received the B.S., M.S., and PhD degrees in Computer Science from Soongsil University, Korea, in 1992, 1996, and 2003. He is currently a professor in the School of Information Engineering, Mokpo National University, Korea. His research interests are the multi-paradigm programming, XML processing, and ubiquitous computing.

Young-Chul Kim received the B.S. degree in Computer Science from Hannam University in 1990, and M.S. and PhD degrees in Computer Science from SoongSil University, Korea, in 1996 and 2003 respectively. During 2004-2005, he stayed in Information Media Technology Research Center, Korea. He is currently a professor in the Department of Electronic Commerce at Yuhan College. His research interests are programming language, compiler, network management, and XML.