UDC 004.4:004.73
Original Scientific Paper

Java Applications Development Based on Component and Metacomponent Approach

Danijel Radošević *danijel.radosevic@foi.hr*
University of Zagreb
Faculty of Organization and Informatics Varaždin

Mario Konecki *mario.konecki@foi.hr*
University of Zagreb
Faculty of Organization and Informatics Varaždin

Tihomir Orehovački *tihomir.orehovacki@foi.hr*
University of Zagreb
Faculty of Organization and Informatics Varaždin

Abstract

Component based modeling offers new and improved approach to design, construction, implementation and evolution of software applications development. This kind of software applications development is usually represented by appropriate component model/diagram. UML, for example, offers component diagram for representation of this kind of model. On the other hand, metacomponents usage offers some new features which hardly could be achieved by using generic components. Firstly, implementation of program properties which are dispersed on different classes and other program units, i.e. aspects, is offered. This implies using automated process of assembling components and their interconnection for building applications, according to appropriate model offered in this paper, which also offers generic components usage. Benefits of this hybrid process are higher flexibility achieved by automated connection process, optimization through selective features inclusion and easier application maintenance and development. In this paper we offer an approach of application development based on hybrid component/metacomponent model. The component model is given by UML diagrams, while the metacomponent model is given by generator scripting model. We explain that hybrid approach on an example of Java Web application development.
Keywords: component model, metacomponent model, web application, Java

1. Introduction

The concept of building software from components has been used for many years. Software is made from components that can be developed or can be bought. This kind of application is more flexible than applications developed using non-component approach because of their plug and play nature.

Components are built to be reusable which makes development of further applications that offer similar functionalities much easier. Another step forward would be metacomponents usage. Metacomponents are used as templates for making final components using an automated process of generation. This means that metacomponents are more general in relation to appropriate components. The generation process of metacomponents depends on their specification, which offer optimization (in relation to standard components) because components generated from metacomponents don't need to contain all possible features of the whole problem domain; i.e. involving of particular feature depends on component specification. Also, metacomponents can be combined in a way that more than one metacomponents can be used in generation of some particular component. Also, some metacomponents can be used in generation of different kinds of components.

So, the metacomponent approach has the potential of making application/component development process more flexible, giving more reusability and some level of optimization in relation to standard component approach.

2. What is a component?

Despite of the fact that components are known for years, when we talk about component based software development (CBSD) we can say that it is a discipline that is still in the process of development [2]. There are several important terms that we can identify in this discipline but the main focus is on a component. A component is a part of a program product. It consists of a group of functionalities that are offered through that component [2]. A component is implemented in some programming language, compiled and as such it represents the black box, that is, the implementation details of a component are not known to its environment.

In order to communicate with its environment, components use one or more interfaces. The interfaces specify the services that a component provides. In most cases interface defines just syntactical aspect of a component (inputs and outputs) and says nothing about semantical aspect. This tells to user very little about what a component really does.

In order to describe functionalities of a component, every component has its contract which defines what component has to be provided with by its environment and what it provides to its environment if the conditions that are specified in the contract are fulfilled. A contract of a component also describes a way of communication/interaction between components in some particular group.

2.1. Types of components

There are 3 main types of components [2]: custom-built components, reusable components and commercial components. Custom-built components are components developed for some particular purpose. Reusable components are components owned by developers of application that have been developed for some other application but can be used for present development. Commercial components are components that are developed for sale on a component market. A UML is mostly used to model a component and to show its structure (Figure 1).

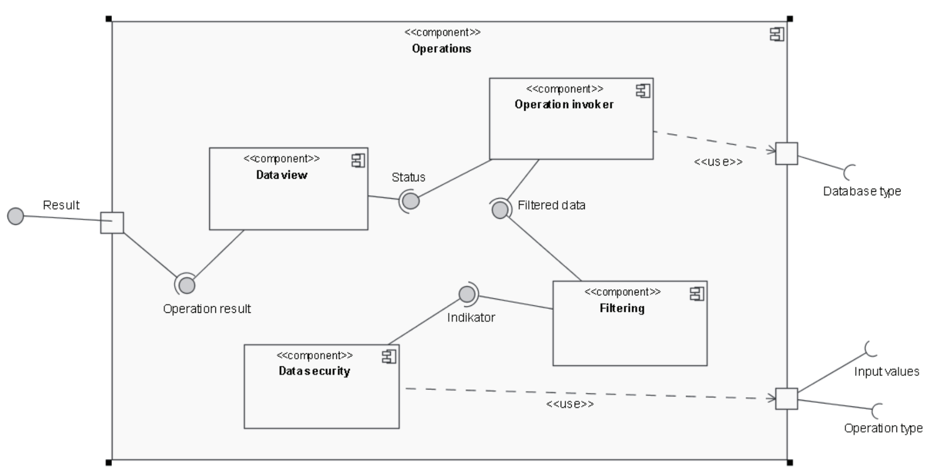

Figure 1. UML model of Java composite component

2.2. Components characteristics

There are some components principles which distinguish them from other programming technologies [2]:

- reusability – the property to use a component developed during one software development process (SDP) in another SDP,
- substitutability - the ability to replace component with alternative implementation of component,
- extensibility – the characteristic which can add new features to individual components or extend one component into two or more components,
- composability – the ability to assemble various component functions in order to satisfy specific user requirements.

Beside the mentioned principles the following are also referred [14][5]:

- executability – component is an executable programming module,
- interface – the property which determines internal running of components,
- source code protection – source code isn't directly accessible to component users,
- interaction between components in order to exchange information,
- flexibility – the property to modify a single component in order to use it in another SDP,
- maintainability – the ability to modify component in order to adapt it to a specific SDP.

2.3. Problems with components

When we consider the process of components assembling we are facing with several problems [10]:

- if we want to utilize a particular functionality we need to identify appropriate component(s),
- there are some gaps between components and desired functionality so we need to specify and resolve them,
- it is necessary to specify interaction between components,
- during the interaction between components in nonlinear systems some emergent behavior can occur.

Because of these problems, we will examine properties which metacomponent approach brings.

3. Metacomponent approach

Metacomponent is, according to Villacis, a "container component that has "inside" knowledge about the connections between components embedded within it" [16]. The main difference between metacomponents and components is that metacomponents are just templates for components, not the whole components that could be included into working applications. So, metacomponents require some automated process to produce components. That process is quite invasive - all changes are hardcoded into program code through the process of generation. The main advantages of this approach are, according to [3]:

- **optimization:** unlike components, which should cover wide area of their problem domain, to fulfill needs of many different applications, metacomponents are pretty "light" - specific properties could be involved by specific needs of particular application (defined in application specification).
- **aspects:** according to Kinczales [7], aspects represent features that are not strictly connected to individual program organizational units like functions or classes, so they can appear within different application parts. During the process of component generation,

particular aspects are integrating with different metacomponents, giving them the features from application specification. It means that some feature have to be defined just once (in the application specification), but dispersed on different application parts. It was shown that UML and other object modeling techniques have significant problems in modeling aspects (i.e. Lee [8]).

- **flexibility:** while components need to be accessed through its public interface, metacomponents allow invasive approach, i.e. can be changed inside. This enables fine adjusting of desired properties.

3.1. Scripting model of generator

Including aspects into generated application requires appropriate connectivity model, which is called, according to Kandé [6], the Join points model. Scripting model of generator [11] is kind of Join points model, where join points are defined as typeless [12] unlike classic object model, where join point are defined as complex connectivity classes. The property of being typeless should make connectivities easier, just like scripting programming languages, which tend to be typeless, and are used for connecting components written in system (structural and object oriented) programming languages [9].

3.1.1. Diagrams of generator scripting model

The scripting model consists of two graphic diagrams (or equivalent textual specifications), so it's simpler in relation to the models based on UML [11]. The first diagram is called the specification diagram and defines the structure of the application specification within the generation system. The specification diagram of Java application for remote database maintaining generator defines features (aspects) which make single application different from other within its problem domain. In the example, specification defines used tables and fields in each table (Figure 2.).

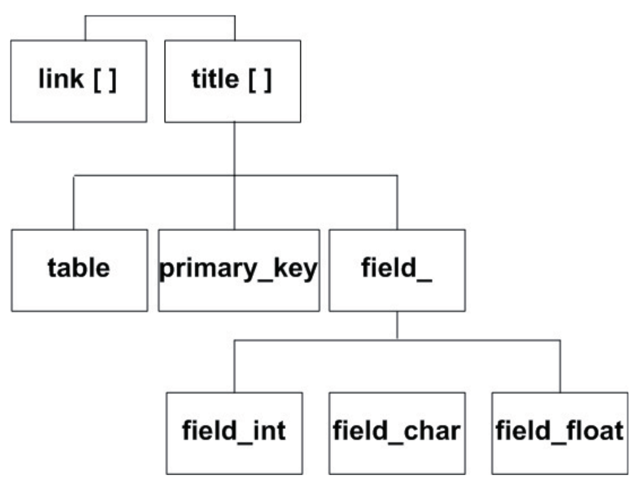

Figure 2. The specification diagram of the example Java application

The generation system generates the application within its problem domain, which is designated by program code templates (metascripts). The connection rules for connecting metascripts to application specification are defined in the second diagram - the metascripts diagram [11]. The metascripts diagram of Java application for remote database maintaining generator defines connections between metascripts and application specification (Figure 3.).

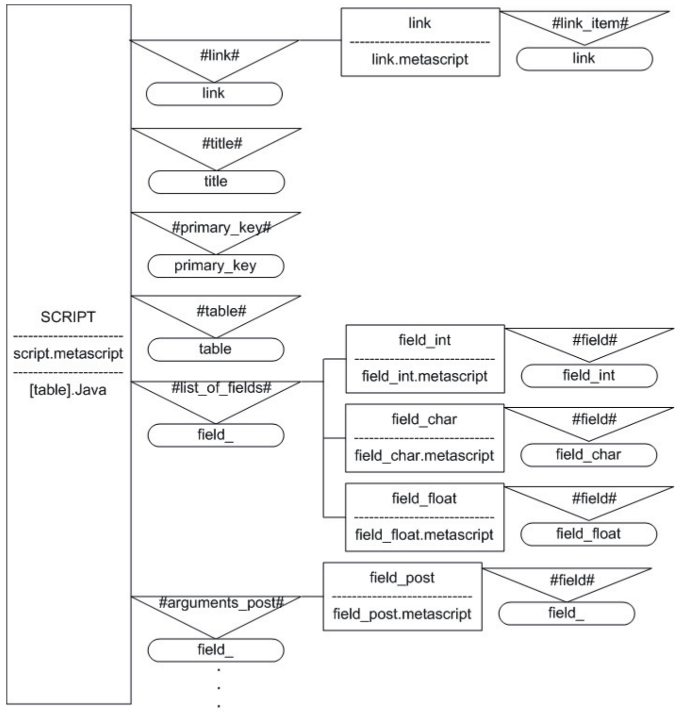

Figure 3. The metascripts diagram of the example Java application

4. Application prototype

In order to develop a suitable generator a desktop/web application prototype was developed using Java technology. This prototype was developed in such way that it consists of all elements and provides all necessary functionality that will be used later in other generated applications. The platform for prototype development was chosen according to some simple guidelines, namely the main reason for choosing Java was its openness and platform independency.

When we talk about Java we talk about programming language but also about platform (a hardware or software environment in which program runs [1]).

When writing java code, all code is first written in plain text files ending with the .java using some editor. The files are called the source files. They are then compiled using java compiler into .class files. The files (.class) contain bytecode that isn't native to computer processor. Bytecode is machine language of the Java Virtual Machine (Java VM [15]). Compiled application is run with the instance of Java VM. Java VM is available on various platforms and that is why java programs are able to run on different operating systems. The process of running Java application is shown in Figure 4.

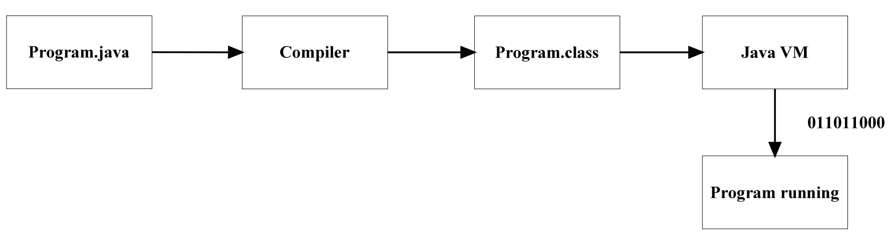

Figure 4. Running Java application

The technology that was used inside of Java is Swing. Swing is a GUI toolkit for Java. It is one part of the Java Foundation Classes (JFC) [4]. Swing includes graphical user interface

(GUI) widgets such as text boxes, buttons, split-panes, and tables. Swing is a platform independent, *Model-View-Controller* GUI framework for the Java system [13]. Swing enables one to develop an application that can be used as a desktop or web application (as an applet inside of a browser). Using this kind of technology a high-level of flexibility was gained.

The database used in this prototype is MS Access database. It was used because of its simplicity but any other database could also be used, without changing any of program code, except database connection string. The database consists of two interlinked tables, called "Students" and "Subjects". The structure of prototype database structure is shown in Table 1 and Table 2.

Attributes	**Data types**
student_id (primary key)	integer
surname_name	varchar
year_of_study	integer
year_of_enrolment	integer

Table 1. Structure of table "Students"

Attributes	**Data types**
subject_id (primary key)	integer
student_id (foreign key)	Integer
subject_name	varchar
semester	integer
teacher	varchar

Table 2. Structure of table "Subjects"

The prototype developed is the base for generating similar and more complex programs. The whole program is written in just one .java file which simplifies generating process. The application prototype implements the following functionalities:

- Inserting new participants
- Updating existing participants
- Deleting existing participants
- Viewing existing participants
- Interlinking of tables using foreign keys

All these functions are implemented on just one screen to simplify the usage of this prototype. Also some other features such as asking confirmation for deleting are also implemented. The screenshot of application is shown in Figure 5.

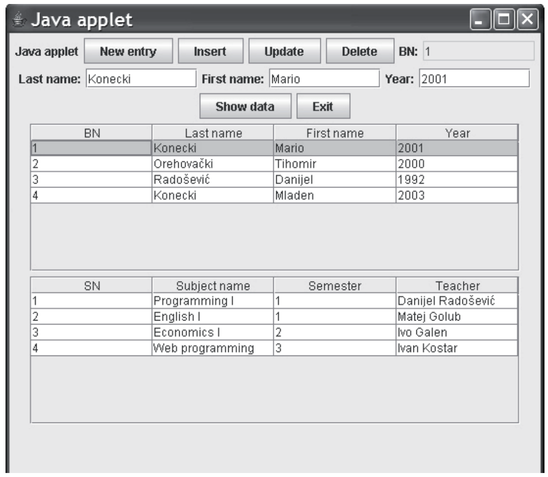

Figure 5. Screenshot of application prototype

5. Generating case

Generating case refers to generating Java applets for database administration (data review; adding. editing and deleting records), according to appropriate specification and program code templates (metascripts).

5.1. Specification

According to scripting model of generator, building an application starts with the specification. For example:

```
link:subjects.student_id->students.student_id

title:Students
table:students
primary_key:student_id
field_int:student_id
field_char:surname_name
field_int:year_of_study
field_int:year_of_enrollment

title:Subjects
table:subjects
primary_key:subject_id
field_int:subject_:id
field_int:student_:id
field_char:subject_name
field_int:semester
field_char:teacher
```

This specification defines tables to be created and maintained (*students, subjects*), with its fields (*id*, *surname and name*, *year of study* and *year of enrollment* for table *students; id, name, semester and teacher* for table *subjects*), primary keys (*student id, subject id*), foreign key (*student id*) and group titles (*Students, Subjects*). These are the features of generated application that varies within its problem domain.

5.2. Metascripts

Metascripts (program templates) define common parts of different applications among its problem domain. Features from specification are connected to metascripts according to the metascripts diagram (Figure 3). In the following example, several features are connected to appropriate metascripts:

```
. . .
 {
 JOptionPane pane = new JOptionPane(
 "#table# with #primary_key# already exists !");
 JDialog dialog = pane.createDialog(new JFrame(), "Data enter failed!");
dialog.setVisible(true);
 #primary_key#_polje.requestFocus();
 #primary_key#_polje.setSelectionStart(0);
 #primary_key#_polje.setSelectionEnd(100);
 }
. . .
```

After connecting to specification (generated parts are bolded):

```
. . .
{
   JOptionPane pane = new JOptionPane(
   "students with that id already exist !");
    JDialog dialog = pane.createDialog(new JFrame(), " Data enter failed!");
 dialog.setVisible(true);
 id.requestFocus();
 id.setSelectionStart(0);
 id.setSelectionEnd(100);
 }
. . .
```

In the example, all tags (marked by # signs) are directly exchanged by values from specification. That is not a case in bit more complex example:

```
public void mouseClicked(MouseEvent e)
{
        int row = table.getSelectedRow();
        int counter=0;
        if ((#table#.getValueAt(row, 0)) != "")
        {       //#primary_key#.setText(table.getValueAt(red,0).toString());//subtemplate
#show_record#
        }
 #primary_key#_polje.setEditable(false);
}
```

After using appropriate specification elements (generated parts are bolded):

```
public void mouseClicked(MouseEvent e)
{
 int row = students.getSelectedRow();
 int counter=0;
 if ((students.getValueAt(row, 0)) != "")
 {
```

```
 id.setText(students.getValueAt(row, counter ++).toString());
surname_name.setText(students.getValueAt(row,counter ++).toString());
year_of_study.setText(students.getValueAt(row,counter ++).toString());
year_of_enrollment.setText(students.getValueAt(row,counter++).toString());
 }
 id.setEditable(false);
}
```

Using of sub templates is defined by lower levels of the metascripts diagram (Figure 3).

6. Combining component and metacomponent approach

Despite of using metacomponent model, like scripting model of generator, the whole approach of application development is hybrid: component and metacomponent approaches are combined.

Why? Looking just application prototype, it could be fully described by its component model, despite the fact that some of the components are generated from appropriate metacomponents. It's not necessary that all of the components have to be generated - some have no features which should be defined in the application specification. The aspiration of generator scripting model is to make application specification as light as possible, so it has to contain only features which have to be different inside the generator problem domain.

In our example of Java database administration application, the component model is given in Figure 6:

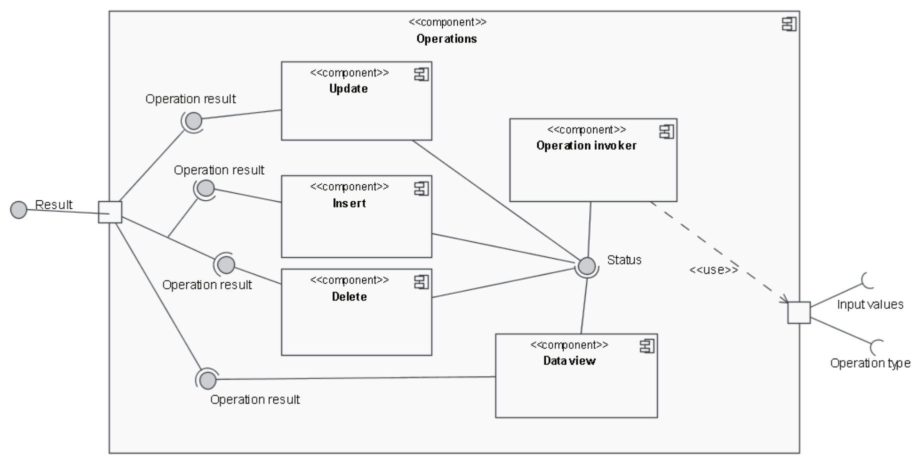

Figure 6. Java database administration application component model

7. Conclusion

In this paper component and metacomponent approach was discussed and an example of Java software generation system built using metacomponents was shown. Appropriate generator was developed using generator scripting model which represents kind of metacomponent model and the main advantages of metacomponent approach, toward to component model have been shown. Regarding the fact that the fully metacomponent approach could be too demanding, we suggest the hybrid component/metacomponent approach. Some of the components are common for all applications inside the generator problem domain and there is no need to generate them from metacomponents. On the other hand, the metacomponent approach includes automatic code generation, some level of optimization (only metacomponents which are really needed according to application specification will be included into final application), and easier further application maintenance through changing its specification and metacomponents.

In our future work we plan to improve the generative application development based on generator scripting model with main accent on following areas:

- problem domain reengineering,
- introducing some new concepts to the scripting generator model, like virtual metascripts, similar to the object model, and
- development of new programming platforms for making generators, except the existing scripting and C++ platform.

References

[1] Arnold, K.; Gosling, J. *The Java programming Language*, Addison-Wesley, 1998.

[2] Crnkovic, I.; Larsson, M. *Building Reliable Component-Based Software Systems*, Artech House, Boston, 2002.

[3] Czarnecki, K.; Eisenecker, U. W. *Generative programming: methods, tools and applications*, Addison-Wesley, 2000.

[4] Flanagan, D. *Java Foundation Classes in a Nutshell: A Desktop Quick Reference*, O'Reilly, 1999.

[5] Gómez-Perez A.; Lozano A. Impact of Software Components Characteristics above Decision-making Factors, *International Workshop on Component-Based Software Engineering (CBSE 2000)*, pp. 15 – 23, Limerick, Ireland, 2000

[6] Kandé, M. M.; Kienzle, J.; Strohmeier, A. From AOP to UML - A Bottom-Up Approach, *1st International Conference on Aspect-Oriented Software Development*, 2002., Enschede, The Netherlands, URL: http://lglwww.epfl.ch/workshops/aosd-uml/Allsubs/kande.pdf

[7] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J. M.; Irwin, J. Aspect-Oriented Programming, *Lecture Notes in Computer Science*, Vol. 1241/1997, pp. 220 – 242, 1997.

[8] Lee, K.W.K. An Introduction to Aspect-Oriented Programming, *COMP 610E 2002 Spring Software Development of E-Business Applications*. The Hong Kong University of Science and Technology, 2002.

[9] Ousterhout J. K. Scripting : Higher Level programming for the 21st Century, *Computer*, Vol. 31, No. 3, pp. 23 – 30, 1998.

[10] Parsons, R. Components and the World of Chaos, *Software*, Vol. 20, No. 3, pp. 83 – 85, 2003.

[11] Radošević, D. *Integration of Generative Programming and Scripting Languages*, PhD thesis, Faculty of Organization and Informatics, Varaždin, Croatia, 2005.

[12] Radošević, D., Kozina, M.; Kliček, B. Comparison between UML and Generator Application Scripting Model, *Proceedings of the 16th International Conference on Information and Intelligent Systems (IIS)*, pp. 115 – 122, Faculty of Organization and Informatics, Varaždin, Croatia

[13] Reenskaug, T. The Model-View-Controller (MVC): Its Past and Present, University of Oslo, URL: http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf, Retrieved May 15 2007

[14] Simão, R. P. S.; Belchior, A. D. Quality Characteristics for Software Components: Hierarchy and Quality Guides, *Lecture Notes in Computer Science*, Volume 2693, pp. 184-206., Springer, 2003.

[15] Stärk, R.; Schmid, J.; Börger, E. *Java and the Java Virtual Machine: Definition, Verification, Validation*, Springer-Verlag, 2001.

[16] Villacis, J. E.; Govindaraju, M.; Stern, D.; Whitaker, A.; Breg, F.; Deuskar, P.; Temko, B.; Gannon, D.; Bramley, R. CAT: A High Performance, Distributed Component Architecture Toolkit for the Grid, *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, Redondo Beach, USA, pp. 125 – 132, 1999.