Java software development using component and metacomponent model

Danijel Radošević, Mario Konecki, Tihomir Orehovački
University of Zagreb
Faculty of Organization and Informatics, Varaždin
danijel.radosevic@foi.hr, mario.konecki@foi.hr, tihomir.orehovacki@foi.hr

Abstract: *Component based modeling offers new and improved approach to the design, construction, implementation and evolution of software applications development. Software components can improve many aspects of software applications development such as functionality, maintainability, usability, etc. Components are used to develop software applications by using some of their services. 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 give an example of Java Web application development based on hybrid metacomponent/component approach.*

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 PNP nature.

Components are highly reusable which makes development of further applications that offer similar functionalities much easier. Another step forward would be metacomponents usage. Metacomponents usage offers automation, optimization and easier maintenance/development and higher flexibility.

Components that are generated from metacomponents consist of just those functionalities that are needed for some particular case, rather than of all functionalities available for some particular component.

2. What is a component?

When we talk about component based software development (CBSD) we can say that it is rather young discipline that is still in the process of development. 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 [1]. 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 ore more interfaces. Interface provides component a way to communicate with its environment, that is, with other components. Interfaces define services that some particular 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 the behavior of a component (what we have to provide to get certain results and which conditions have to be met in order to get the right results). 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 [1]: custom-built components, reusable components and commercial components. Custom-built components are components developed for some particular purpose (e.g. Figure 1.). 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.

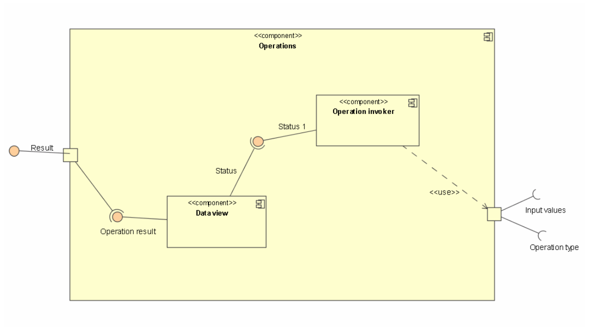


Figure 1. UML model of Java component

2.2. Components Characteristics

There are some components principles which distinguish them from other programming technologies [1]:

- 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 [11][3]:

- 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 [8]:

- if we want to implement desired 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 Villacıs, 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 [2]:

- **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 [5], 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. 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 modelling techniques have significant problems in modelling aspects (i.e. Lee [6]).
- **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 [4], the Join points model. Scripting model of generator [9] is kind of Join points model, where join points are defined as typeless [10] 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 [7].

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 [9]. 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 it's problem domain. In the example, specification defines used tables and fields in each table (Figure 2.).

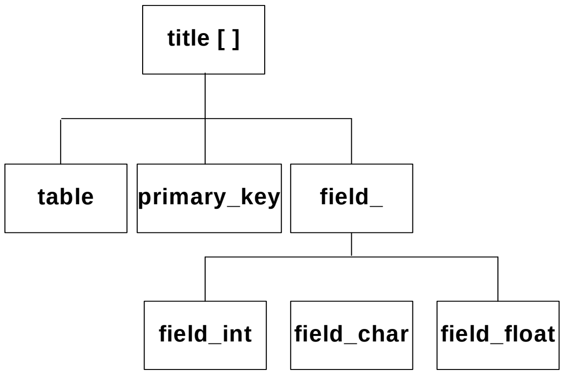


Figure 2. The specification diagram of the example Java application

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 [9]. The metascripts diagram of Java application for remote database maintaining generator defines connections between metascripts and application specification (Figure 3.).

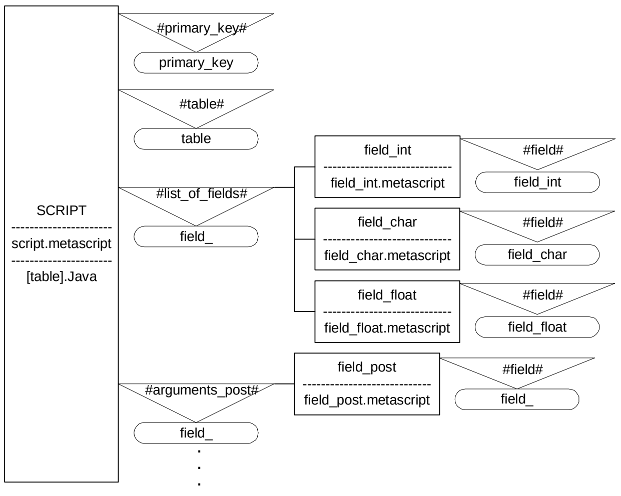


Figure 3. The metacsripts diagram of the example Java application

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 [12]).

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 javac 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 [13]). 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.

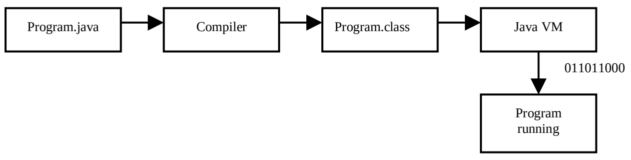


Figure 4. Running Java application

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) [14]. 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 [15]. 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 just one table called "Participants". The structure of prototype database table is shown in Table 1.

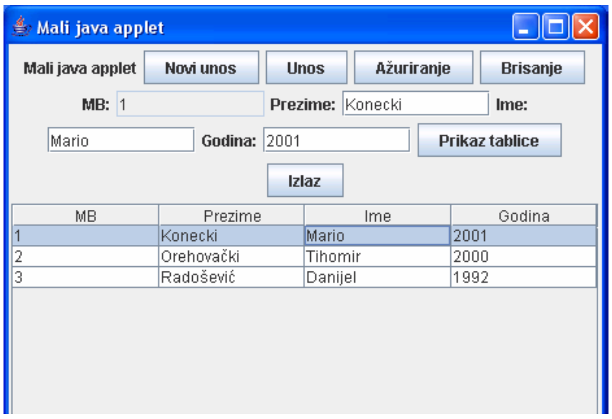
Attributes	Data types
student_id (primary key)	integer
surname_name	varchar
year_of_study	integer
year_of_enrolment	integer

Table 1. Structure of prototype database table

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 functionality:

- Inserting new participants
- Updating existing participants
- Deleting existing participants
- Viewing existing participants

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.



MB: 1 Prezime: Konecki Ime: Mario Godina: 2001

MB	Prezime	Ime	Godina
1	Konecki	Mario	2001
2	Orehovački	Tihomir	2000
3	Radošević	Danijel	1992

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:

```
title:Students
table:students
primary_key:id
field_int:id
field_char:surname_name
field_int:year_of_study
field_int:year_of_enrollment
```

This specification defines table to be created and maintained (*students*), with its fields (*id*, *surname name*, *year of study* and *year of enrollment*), primary key (*id*) and group title (*Students*). 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).toStr
ing());//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,cou
nter ++).toString());
year_of_study.setText(students.getValueAt(row,count
er ++).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:

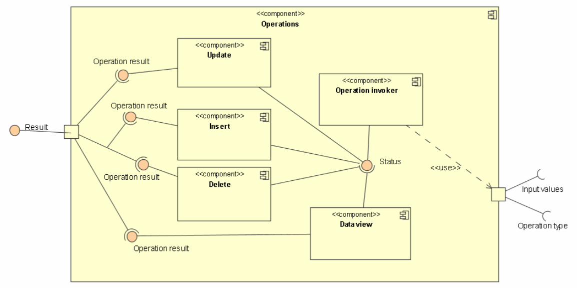


Figure 6: Java database administration application component model

7. Conclusion

In this paper component and metacomponent approach was compared on an example of Java software generation system. Appropriate generator was developed using scripting generator 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, used approach was really hybrid. Some of the components are common for all applications inside the generator problem domain and there is no need to generate them from metacomponents. That means that whole software development system could be defined by both component and metacomponent model, keeping advantages from both of them.

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.

8. References

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

[2] Czarnecki, K.; Generative Core Concepts - Generative Domain Model, Program-Transformation.Org, 2002., http://www.program-transformation.org/Transform/GenerativeCoreConcepts

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

[4] 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, http://lglwww.epfl.ch/workshops/aosd-uml/Allsubs/kande.pdf

[5] Kinczales, G.; Lamping, J.; Mendhekar, A.; Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. Aspect-Oriented Programming". In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997. http://citeseer.nj.nec.com/ kiczales97aspectoriented.html

[6] Lee, K. W. K.; An Introduction to Aspect-Oriented Programming, COMP610E: Course of Software Development of E-Business Applications (Spring 2002), Hong Kong University of Science and Technology, 2002.

[7] Ousterhout J. K.; Scripting : Higher Level programming for the 21st Century, IEEE computer magazine, march 1998.

[8] Parsons, R.; Components and the World of Chaos, IEEE Software, Vol. 20, no. 3, pp. 83 – 85, 2003.

[9] Radošević, D.; Integration of Generative programming and Scripting Languages, doctoral thesis, Faculty of Organition and Informatics, Varaždin, Croatia, 2005.

[10] Radošević, D., Kozina, M.; Kliček B.; Comparison Between UML And Generator Application Scripting Model, Conference proceedings of Information and Intelligent Systems 2005 (IIS 2005), Fakultet organizacije i informatike, Varaždin, 21.-23.09.2005.

[11] Simão, R. P. S.; Belchior, A. D.; Quality Characteristics for Software Components: Hierarchy and Quality Guides, Component-Based Software Quality, Volume 2693, pp. 184-206., Springer Berlin / Heidelberg, 2003.

[12] Sun Microsystems; About the Java Technology, http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html, Sun Microsystems, Inc., 2006.

[13] Sun Microsystems; Java Virtual Machines, http://java.sun.com/j2se/1.4.2/docs/guide/vm/index.html, Sun Microsystems, Inc., 2002.

[14] Sun Microsystems; Java Foundation Classes, http://java.sun.com/products/jfc/reference/, Sun Microsystems, Inc., 2007.

[15] Sun Microsystems; Model-view-controller, http://java.sun.com/blueprints/patterns/MVC.html, Sun Microsystems, Inc., 2002.

[16] Villacıs, J. E.; The Component Architecture Toolkit, Indiana University, Department of Computer Science, 1999., http://www.extreme.indiana.edu/ cat/papers/discat.pdf