Use Case Components for Interactive Information Systems

Eliezer Kantorowitza and Lyakas Alexandera

aComputer Science Dept., Technion—Israel Institute of Technology, Haifa 32000, Israel

Specification-oriented components are components designed to facilitate the implementation of a system from its specifications. An earlier study has shown cases where such components enabled information systems to be implemented with considerably less code than when implementing them with components, designed by a typical object-oriented design approach. This study goes a further step by using a higher level of abstraction, where the essence of an information system is assumed to be the flow and processing of data. The components based on this abstraction attempt to hide the kinds of code, that appear in all interactive Web-based information systems and that are not data flow or data processing code. Based on this approach an experimental framework called WebSI has been developed. WebSI provides components, that strive to hide the code for the construction of the graphical user interface, the database access code and the Web-related code. WebSI was designed to facilitate the manual translation of natural language use case specifications into Java code. In the experiments these components enabled the construction of information systems with a modest amount of code. The similarity between the WebSI-based Java code and the English language use case specification facilitated both the coding of the specification and verifying, that the code implements the specifications correctly.

Keywords:
component, specification-oriented, use case, information system, user interface, World Wide Web

1. Introduction

A software component may be defined as a module of software, that is designed such that it may be reused in many different systems. The design of components that meet this requirement has proven to be a difficult task. These difficulties may be alleviated by designing components, that are limited to a particular application domain and to a particular method of combining the components into a piece of software. Such a set of components and the corresponding construction method is often called a component framework.

One of the measures of quality of a framework is the developer effort required to design, implement and test a piece of software with the framework, i.e., the development costs. The term development is employed in this paper for the development of software in the application
domain of the framework. Another quality measure of a framework is the developer effort required to extend the software produced with the framework at a later point of time, i.e., the *extension costs*. This paper focuses on the design of frameworks with purpose of achieving low implementation and extension costs. Other aspects of component design, e.g., how to connect components of different frameworks in a single system, are not considered in this paper.

One approach in framework design is to use a classical object-oriented design approach, i.e., to construct a component for each one of the important objects of the application domain. An example of such a framework is the Java’s Swing package for graphical user interface (GUI) construction. This package is composed of components for such GUI controls as buttons, menu entries and text boxes. A GUI is constructed by combining these components in a special way. The construction involves the specification of the size of the controls and their layout on the screen. The programmer must also code the operations carried out by the software, when a particular GUI control is operated by the user. Programming at this level of detail can be quite labor intensive.

Another approach in framework design is the *specification-oriented* approach, suggested in [3], where the components are designed to enable a direct coding of the system from its specifications. The possibility of deriving an implementation of a system directly from its specifications was demonstrated for reactive systems, where a state chart specification of such a system can be executed. The developer of such a system has thus only to specify it by its state chart, and need not invest further time in designing and implementing the system [4]. The state chart can be executed, because it explicitly specifies all the computations in the system. This study considers large complex information systems, where it is difficult to produce a complete specification of the entire system, as done in the state chart approach. A common industrial method to manage this high complexity is to develop such system through a software development process (often shortened to *software process*). Each phase of such a software process focuses on one aspect of the development, such that the developers can concentrate on doing it right.

A number of different kinds of software processes have been designed over the last decades. This research was specially influenced by [9], which is the introduction of the *use case* concept and its application in a software process. The use case concept and additional important concepts from other software processes were later elaborated into the Unified Software Development Process (USDP) [1], which was developed in connection with the popular standardized Unified Modelling Language (UML) [2].

The early software processes employed a waterfall model, where an entire software system is developed in one pass. It was, however, observed that the insight gained during the later implementation phase sometimes suggests significant improvements to the specification developed at an earlier phase. Some software processes therefore permit iterations to incorporate such improvements. Other processes, like the spiral process, divide the software into a number of parts, that are developed one after the other, such that insights gained with early parts may be exploited when developing the later parts. Both the iterational and the incremental processes employ the waterfall model during an iteration or the incremental pass.

The different phases of a software process may be divided into two major stages. The first, *requirements and specification* stage, comprises of requirements elicitation and a detailed specification of the system to be constructed. UML and USDP employ use case specifications. A use case is a single application of a system. Use cases are often described in a natural language,
e.g., English, and are accompanied by some description of the graphical user interfaces (GUI). An example of a use case is the use of a sales offers management information system to add a sales offer of a particular supplier for a particular item on sale:

- The system presents the set of items for sale, for which the supplier has not yet given offers. For each item, its name and description are presented.
- The supplier selects a single item to give an offer for.
- The supplier specifies the price.
- The supplier approves.
- The system registers the new offer and presents a success message.

The above use case specification does not tell how the system locates the items for which the supplier has not yet given offers. The specification also provides no information on where and how the details of the new offer are stored. So as opposed to a state chart specification, a use case specification is not required to specify how the system computes its outputs from the user’s inputs. It is essentially only a specification of user inputs data and the corresponding system outputs data.

The use of natural language and GUI descriptions enable domain experts and user interface experts, that may not be familiar with formal notations, to validate the use case specification. The purpose of this validation by both domain and user interface experts is to ensure the usability of the system. Usability expresses both the extent to which the system provides the services that the users need, and whether it enables the users to accomplish their work with a minimum of human effort and in a pleasant way. Usability is possibly the most important system property as it may be determining for whether the software will be purchased. The product of the first stage of the USDP process is thus a use case specification of the system, that is validated for its usability.

In the second stage of the software process, the construction stage, a system is developed, that implements the use case specifications, which were developed in the first stage of the process. At this stage the method for computing the system outputs from the users inputs and database values is developed. Usually the developer has to select one of a number of different possible methods for this computation. For the selected computation method the developer designs and codes the software, that implements that method. The construction stage involves a verification, that the code of the developed system, implements the use case specifications precisely. Since the use case specifications were validated for their usability, the verified code will retain this usability.

The second, construction stage of the software process is usually labor intensive, as it involves analysis of the specifications, system design, implementation and verification. In order to reduce the high costs of these construction activities one may implement the system with the help of high-level reusable software components. Such a set of components is suggested in [3], where the concept of specification-oriented components was introduced. The purpose of this approach is to facilitate the verification of the code. Specification-oriented components should enable a translation of the natural language use case specifications into high-level code, that resembles the specification, such that the equivalence between the code and specification is relatively easy to establish.
The approach was tested with the experimental system, named SI (Simple Interfacing) [3], for construction of interactive information systems. In the experiments, the English language use case specifications where manually translated into SI-based Java code. Typically, an English statement in the use case specification was translated into 2–3 Java statements, that resembled the specification to a degree that made it relatively easy to verify, that the code corresponds to the specification. One system was implemented twice: once using the Java’s GUI Swing framework and the Java’s database connectivity package JDBC, and the second time using the components of SI. There were 2.4 as many lines of code in the Swing and JDBC-based implementation than in the SI-based code. This suggests that using components that model use cases, as done in SI, saves coding, as compared to using components that model the objects of the domain in a classical object-oriented design, as is done in the Swing framework. This suggestion supports the observation made by [10], that a classical object-oriented component design does not necessarily produce the most appropriate components. The remarkable code savings obtained with SI were achieved mostly by hiding much of the GUI and database access code in the components of SI. A later design of SI, called SI+, is reported in [5,6].

Based on the experience with SI and SI+ we have developed an experimental system, called WebSI (Web Simple Interfacing), for specification-oriented construction of Web-based information systems. WebSI is a set of Java components designed to facilitate the implementation of Web-based information systems, specified by their use cases, which are assumed to be written in English. The translation of the use case specifications into Java is done manually by the developer, that implements the information system. WebSI may thus be employed for the second (construction) stage of the software process, where it is required to implement a system from a given use case specification.

In the following sections we discuss the principles employed in the design of WebSI, as well as the software engineering properties of information systems constructed with it.

2. The Architectural Principles of the Experimental System WebSI

The usage model of a WebSI-based information system is that the user initiates a use case and inputs (enters) required data. The use case performs the needed computation and outputs (displays) the results to the user and/or stores them in the database. It is assumed that all data needed for the computations are either entered by the user or stored in the database, which may therefore be considered as the state of the information system. Similar to many current information systems, the use cases are assumed to access the database through transactions, designed to retain the consistency of the database.

Since a use case is assumed to employ only the user and the database data, the use case only interfaces with the database and the user, but not with the other use cases. This architecture is illustrated in Figure 1. The developer of a particular use case need therefore usually not know anything about the other use cases, which means that a particular use case is not coupled to any other use case components. Therefore, the use case developer needs usually only know the use case specification and the structure of the database schema.

WebSI assumes, that the essential activities of an information system are the moving and processing of data. The design of WebSI therefore attempts to hide all the code, that is not related to the moving and processing of data. This hiding enables the developer of the use case specification and the programmer, that translates the use cases into Java code, to focus on the
flow and processing of the data.

Similar to many information systems, WebSI employs a single relational database, that stores the state of the information system.

When employing WebSI, each use case is implemented by a separate class (called the use case class), which is derived from the WebSI $\text{UseCase}$ class. Each use case class has a number of different interaction methods, that contain the translation of the English use case specification into Java, employing WebSI components. The interaction methods implement the interaction between the user and the system, as specified in the English use case description. Each use case class must implement an interaction method named $\text{start}$, which is invoked by WebSI when the use case execution starts. Other interaction methods are invoked according to the user actions and the use case specification.

A use case implementation may, besides the WebSI components, employ components from other sources. For example, an external $\text{CreditCardValidator}$ class may be used to validate credit card details, when processing a customer’s order in the sales offers management information system, mentioned above. In a WebSI-based information system each use case is thus implemented by a class derived from the $\text{UseCase}$ class and possibly a number of components from other sources.

Since the WebSI-based application is essentially a set of use case classes, we obviously need a kind of a component, that “glues” all the use cases together into one application. For that purpose, WebSI provides a ready-made component, called a use case displayer [6]. The use case displayer’s primary function is to present all the use cases of the application to the user and to enable the user to execute them, as illustrated on Figure 1. Figure 6 shows a use case displayer, that is visualizing the “Add Offer” use case. Figure 8 shows the same displayer, that is visualizing the “Add Item” use case, that enables a supplier to add a new item for sale.

In the following sections we discuss how the emphasis on the flow and processing of data
influenced the design WebST’s GUI services, database access services and its Web services.

3. GUI Facilities of WebSI

From the WebSI perspective, the essence of the user interface is the data, that the user inputs to the system, and the data, that the system outputs to the user. The graphics of the GUI are thus only considered as a means to facilitate the work of the user. Therefore WebSI hides most of the code, that constructs and manages the graphics. This hidden code is responsible for selecting the types of user interface controls to be employed, for the detailed dimensioning of these controls, their layout on the screen, their colors and the employed fonts. The code that detects user inputs at the different controls and initiates the required activities is also hidden. The automatic generation of the GUI employs a WebSI component, called interaction style [7]. The interaction style specifies the kinds of controls to be employed in different situations, the geometrical properties of these controls, their color, their layout on the screen and so on. The developer of an information system has only to select one of a number of different interaction styles, offered by WebSI, and the system will automatically produce a GUI using the selected interaction style. Different interaction styles may be used in different parts of the GUI.

To illustrate the properties stated above let us see an example. Consider again the sales offers management information system, whose database has the relations (tables) “Item”, “Supplier” and “Offer”. The domains (columns) of these tables are listed below:

- Item (id, name, description)
- Supplier (id, name, city, address, description, password)
- Offer (offerid, itemid, supplierid, price)

The columns, whose names are underlined, are the primary keys of the tables.

The information system has a use case called “Edit Offer”, that enables a supplier to change the price of an offer or to remove an offer. Below is a part of the English language specification of this use case:

- The system presents the set of all the currently valid offers of the supplier that executes the use case. For each offer, the item name, item description and price are displayed to the supplier.
- The supplier selects one offer in the set.
  - The supplier specifies a new price for the selected offer.
  - The supplier approves the new price.
  - The system updates the price of the selected offer and presents a success message.

OR

- The supplier instructs the system to delete the selected offer.
- The system deletes the selected offer from the database and displays a success message.
1. `DefaultPane pane = new DefaultPane(this);`
2. `ResultSet offers =`  
3. `read("SELECT offer.offerid, item.name, item.description, offer.price " +`  
4. `FROM offer, item " +`  
5. `WHERE offer.itemid=item.id AND offer.supplierid=" + supplier_id);`  
6. `int[] visible_cols = {2,3,4};`  
7. `int return_values_column = 1;`  
8. `pane.inputChoiceOneOutOfN("offer_id", offers,`  
9. `visible_cols, return_values_column,`  
10. `"Choose the offer you wish to edit: ");`  
11. `pane.inputText("new_price", "Give new price for the selected offer: ");`  
12. `pane.inputAction("updateOffer", "Update the selected offer!");`  
13. `pane.inputAction("deleteOffer", "Delete selected offer!");`  

Figure 2. The Implementation of the Beginning of “Edit Offer” Use Case

Note that the above description contains no GUI details, as those will be determined automatically by `WebSI`.

The programmer translates the above English specification into code of the `start` method of `EditOffer` use case class, as shown on Figure 2.

Line 1 creates a `pane` object, that represents an abstract area, where the interaction between the user and the system takes place. The pane is an instance of the `DefaultPane` class, that implements the default `WebSI` interaction style.

Lines 2–5 perform an SQL SELECT query, that retrieves all the offers of the supplier, that is currently logged in. The query returns a table (into a local variable, called `offers`), that has four columns.

In lines 8–10 the `inputChoiceOneOutOfN` method is invoked on the `pane` object. This method produces the user interface, that enables the user to choose exactly one row out of the `offers` table, produced by SQL query in lines 2–5. Note that the method only specifies the `selection semantics`, but it does not specify which concrete user interface controls should be used to implement this semantics. The programmer specifies (in line 6), that the second, third and fourth columns of a table (item name, description and price) will be visible to the user. The programmer also specifies (in line 7) the `return values column`, which is the column, whose values will serve as return values: e.g., if the user chooses the 3rd row of the table, the programmer will later retrieve the 3rd value of the return values column. The “`offer_id`” string serves as an ID, that will be used by the programmer later to obtain the user’s choice:

```java
...  
    int chosen_offer_id = getInt("offer_id");  
...```

Here programmer retrieves the user’s choice, by getting the value in the `return_values_column`, in the row that has been selected by the user.

Additional input methods used by the programmer are: `inputText`, that enables the user to input some text and `inputAction`, that enables the user to initiate an action in the system. The `inputAction` method receives as an argument the name of the interaction method, that will be invoked when the user initiates this action. Again, it should be noticed, that the pro-
Figure 3 presents the user interface produced by the code on Figure 2. Note, that the interaction style has used radio buttons to implement the single selection semantics and hyperlinks for the action semantics.

If the programmer is unhappy with the appearance of the produced GUI, she may exchange the employed interaction style with another one, as demonstrated on Figure 5. Here we consider a further use case, called “Select Offers”, that enables the customer to browse the offers of the various suppliers. The use case begins with the customer selecting a single city, a single item and approving her choice. Figure 4 presents the code, that implements the beginning of this use case. Here the programmer uses an overloaded version of inputChoiceOneOutOfN method, that enables the user to choose a single value out of the set of values, that comes from one of the columns of the table, which is a result of SQL SELECT query.
Figure 5. Different Implementations of the Same Input Semantics

Figure 5 presents two user interfaces, that can be produced by the code on Figure 4, while employing different interaction styles. The interaction style on the left used single-selection list boxes for the single selection semantics and a push button for the action semantics. The interaction style on the right used combo-boxes to implement the single selection semantics and a hyperlink to implement the action semantics.

Table 1 lists some of the WebSI input and output methods.

As we already mentioned, the user interfaces of the various use cases are made accessible to the user through the use case displayer component. The use case displayer can also be thought of as an interaction style component, since each displayer has a different appearance. Figures 6 and 7 demonstrate two different use case displayers, that are currently visualizing the “Select Offers” use case. Choosing the desired displayer is done during the deployment of the information system, and requires no changes to the client programmer’s code.

4. Database Access with WebSI

We have assumed that the essence of an information system is the moving and processing of data. Moving of data to and from the assumed single database is therefore anticipated to be a frequent operation, that deserves to be supported by WebSI. Moreover, it is expected that a considerable part of the computations in an information system may be accomplished by the powerful SQL language. WebSI permits therefore the use of SQL to manipulate database data. The programmer need not know the actual location of the database, as it is specified during deployment. The various technical activities, such as connecting and disconnecting from the database, managing database transactions, recovering from different database-related errors are
<table>
<thead>
<tr>
<th>Input / Output Method</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Display some text to the user</td>
</tr>
<tr>
<td>output</td>
<td>Present to the user the results of SQL SELECT query (all or some of the columns are visible)</td>
</tr>
<tr>
<td>inputText</td>
<td>Enable the user to input some text</td>
</tr>
<tr>
<td>inputLargeText</td>
<td>Enable the user to input some long text</td>
</tr>
<tr>
<td>inputAction</td>
<td>Enable the user to initiate an action in the system</td>
</tr>
<tr>
<td>inputChoiceOneOutOfN</td>
<td>Enable the user to choose a single value out of the set of values</td>
</tr>
<tr>
<td>inputChoiceOneOutOfN</td>
<td>Enable the user to choose a single row out of a table</td>
</tr>
<tr>
<td>inputChoiceZeroOrMoreOutOfN</td>
<td>Enable the user to choose zero or more values out of the set of values</td>
</tr>
<tr>
<td>inputChoiceZeroOrMoreOutOfN</td>
<td>Enable the user to choose zero or more rows out of a table</td>
</tr>
<tr>
<td>inputActionChoiceOneOutOfN</td>
<td>Enable the user to choose a single value in a table; the user’s choice itself initiates an action in the system</td>
</tr>
<tr>
<td>inputDate</td>
<td>Enable the user to specify a date</td>
</tr>
<tr>
<td>inputTime</td>
<td>Enable the user to specify time</td>
</tr>
<tr>
<td>inputBoolean</td>
<td>Enable the user to answer a yes-no question</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some of the WebSI Input-Output Methods (some are overloaded)</td>
</tr>
</tbody>
</table>

---

Figure 6. A Use Case Displayer
Figure 7. A Use Case Displayer

Figure 8. A Use Case Displayer
handled automatically by WebSI.

4.1. Retrieving Data from the Database

WebSI provides a method read, that performs an SQL SELECT query and returns the resulting table to the programmer. The signature of this method is:

```java
java.sql.ResultSet read(String SQL_SELECT_query);
```

where ResultSet is an interface of the Java language java.sql package.

Individual values of a table returned in the ResultSet can, of course, be inspected with the help of the ResultSet methods. However, in most cases this is not required, since many WebSI input and output methods are able to work with ResultSets directly. We already saw an example for that on Figure 2, in the code that created the user interface on Figure 3. In this example, the possible choices for the user are produced directly from an SQL SELECT query, i.e., from the ResultSet. Another example is the output method, that presents the results of a SELECT query to the user:

```java
void output(ResultSet table, String caption);
```

4.2. Modifying the Database

Similar to retrieving data, there is a single method, called write, that enables the programmer to modify the data in the database:

```java
int write(String SQL_QUERY);
```

The parameter to this method is an SQL INSERT, UPDATE or DELETE query, and the return value is the number of rows affected by the query.

4.3. Transactions and Database Constraints

As we already mentioned, WebSI employs database transactions to preserve the consistency and integrity of data. A new transaction is started on-demand, when the code of an interaction method issues a first SQL query. When the interaction method ends, WebSI tries to commit the transaction, i.e., to commit all the queries the programmer has issued in the code of the interaction method. In case the transaction cannot be committed, WebSI rolls it back and sends an error message to the user.

Database systems are capable of checking various integrity constraints on the database schema, such as PRIMARY KEY or FOREIGN KEY constraints. In case such a constraint is violated, WebSI conveys the violation to the programmer, so that an appropriate action can be coded.

5. Evaluation

Our three different experimental systems—SI, SI+ and WebSI—have been used in several student projects. The projects included systems for air travel planning, football games scheduling, theater events management, automatic assignments of lecturers to courses, and the discussion forums website. The following discussion of WebSI is based on the experience with these projects.

WebSI was designed to provide a higher level of abstraction than its two predecessors—SI and SI+. This is manifested, for example, by WebSI automatic selection of the kinds of controls to be employed in the GUI, which contrasts with SI and SI+, where the developer had to specify the kind of each single control in the GUI. The developer working with WebSI has, on the other hand, only to specify a single interaction style for each part of the GUI. It may
be expected, that using the same single interaction style in each part of the GUI will produce a more uniform appearance than a hand-coded GUI. It was observed in the student projects, that the the ready-made interaction styles that came with the system were satisfactory in most situations [3]. There were only few cases were a new interaction style had to be developed in order to meet special needs [3]. Figures 3, 5, 6, 7, 8 illustrate what is possible with the currently available WebSI interaction styles.

The developer working with WebSI controls the appearance of the GUI by selecting appropriate interaction styles for the different parts of the GUI. If the appearance is not satisfactory, the employed interaction styles may be exchanged with other, hopefully more appropriate interaction styles. Exchanging the interaction styles requires only minimal effort. The developer may therefore try a number of different interaction styles until a satisfactory GUI is achieved. If this is not possible, a new interaction style may be tailored to this case. Software manufacturers may develop interaction styles that meet their GUI standards.

The kinds of user interface controls employed, together with their graphical and geometrical properties are not specified in the client programmer’s code. The WebSI-based code is thus platform-independent. It has therefore the potential of being used on both standard Web browsers, as well as on mobile, Web supporting phones. The same WebSI-based code employing different interaction styles, adapted for the very different GUI capabilities of these platforms, will probably produce very different GUI's.

The usage model and the information system model employed, produced the WebSI-based information system architecture, illustrated on Figure 1. In such a system a use case usually interfaces only with the user and the database, but not with other use cases. The developer of a use case need therefore not know anything about the other use cases, but is required to understand the use case specification, the database schema and the semantics of the WebSI methods. The effort required for extending the information system with a new use case is thus independent of the number of already existing use cases, i.e., the extension complexity [8] of the architecture is $O(1)$. This property is useful for extending the information system to meet needs, that were not known when the system was designed.

The achievement of the $O(1)$ extension complexity is based on two assumptions. The first is that it is possible to design the use cases such that they need not interface with other use cases. This was possible in all our projects, but we do not know if it is true in general. The second assumption is that the database schema is not changed in such a fundamental way, that it cannot be used by both the old and the new use cases. Such fundamental database schema changes are probably very rare. This is supported by the observation of [9], that the schema of database changes much slower than their applications.

Several aspects of WebSI-based information systems development facilitate ensuring their correctness. The experience from the projects supported our expectation about the relative ease of establishing the equivalence between the English language use case specification and the WebSI-based Java code. This facilitated the verification of the code. It was also relatively easy to correct the code when needed.

Another observation was that, as expected, the use of the high-level WebSI components considerably reduced the implementation effort of the students. This enabled the students to invest a larger proportion of their time into requirements elicitation and use cases design. The result was that the developed information systems had generally a better usability than we have experienced in previous projects, where the students used the traditional waterfall development
6. Conclusions

This study suggests an abstract information system model, where only the flow and processing of data are considered. A principal achievement of this research is the design of a framework of specification-oriented components, that to some extent implement this model. The components achieve this by hiding kinds of code, that are required in all interactive information systems, and are not related to the flow and processing of data. This involves the code for construction of the GUI, the database access code and the Web-related code.

WebSI has not been tried on large real-life projects. We believe, however, that since it is based on proven information system principles, it should be applicable for large systems.

The current available interaction styles were useful in the projects. Based on the gained experience it is expected that interaction styles, that meet the standards of a particular software manufacturer, may be developed.

The use of the components of the experimental system enabled the implementation of projects with a modest effort. The saved time may be invested in improving the specifications, with the goal of producing information systems with high usability level.

REFERENCES