I.J. Information Engineering and Electronic Business, **2014, 2, 41-46**
Published Online April 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2014.02.05

Detecting Return Value Mismatch during Component Adaptation with Concern of System Performance

Aisha Mohammed Alshiky, M. Rizwan Jameel Qureshi
Department of Information Technology, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
aalshikey@yahoo.com, anriz@hotmail.com

Abstract—Component adaptation becomes a critical problem in component-based software engineering (CBSE). This problem appears during assembling and reusing of components into new system. The interoperability among components needs to use adaptation technique to solve this problem. Usually, there are mismatches between interfaces of reusable components. This research will focus on detecting fully semantic component interface mismatch by proposing a solution including return value match next to other match such as operation name, method of operation and parameter type match. This research just explains return value mismatch that are not considered in other solutions. The proposed solution also concerns about system performance that is neglected in previous solutions by proposing a 'Detector' tool that is responsible to assign and delete unwanted functions from reusable component before integration.

Index Terms—Component adaptation, fully semantic mismatches, return value mismatch.

I. Introduction

Software component adaptation is a crucial problem in component-based software engineering (CBSE). The main advantages of CBSE are time of development will decrease while quality will increase. Subsequently, CBSD based application development goes through multiple step which are selection, adaptation, and assembly process for components, rather than starting with software development from the scratch [1]. Software developers like to use off the shelf components having high quality and reliability to cut down development time. Each component provides different functions to meet different requirements.

As a result of the incompatibilities among components the need of adaptation methods exists. The commons methods provided to solve such problems are: changing the existing component or system during integration; modify the purchased component as developer wishes and placing an adaptor between the reusable component and system [1]. This paper is addressing the adaptor problem to detect mismatching between requirements and components. There are two classification of adaptation of existing reuse unit that are white-box and black-box. White-box adaptation needs to understand internal component while the black-box adaptation needs to understand just interface of subject of reuse [1]. This research uses black-box adaptation to focus on detecting component interface mismatch problem. Many research papers are written to solve this problem but most of them are inefficient regarding system performance and fully semantic match. Most of studies supported only operation name, method of operation and parameter type mismatch. While the return value mismatch are not considered. This paper proposes a solution that includes return value mismatch next to operation name, method of operation and parameter type mismatch to support. The proposed solution uses semantic match. A tool is proposed naming 'Detector Tool' that assigns and deletes unwanted functions from reusable component to save system performance.

The rest of the paper is organized as follows. Section 2 provides the literature review and limitations. Section 3 describes the problem and proposed solution. Validation of the proposed solution is illustrated in section 4.

II. Related Work

Component based software engineering (CBSE) concept is not new and huge numbers of off-the-shelf components are exactingly available. Each component provides different functions. Mismatch problem is often faced by the developers during assembling and reusing of existing components. Integration among components needs adaptation to solve this problem. There are a lot of studies which is performed to propose solutions for this problem.

One of the mismatch solutions used behavior approach [2]. This solution depends on adaptor concept. The adaptor is used for synchronous interaction between components. The system computes a behavior protocol of adaptor to coordinate the interaction of components. This solution needs to remember the order of message and make them reorder [2]. It is proposed that binary component adaptation techniques are suitable for effective adaptation of components [3-4]. The main drawbacks of these solutions are platform dependency. The static information of component structure and the classes are required to analyze the existing binary component. The portions necessary for adaptation are identified after understanding this information. A solution is proposed for the integration of new requirements with the existing components [5]. Integration code is either reused and adapted, or completely optimized depending on the system's specification. Another study focuses on detection of mismatch [6]. The previous adaptation models focus only on message name mismatch while this study considers data type mismatch in addition to message and parameter mismatch. This study suggested a solution focusing into parameters data types [6].

Test-driven prototypes can't find semantics of matching components [7-8]. Adaptation is difficult to specify at run time of system [9]. An approach is suggested using unified meta model of each component [9]. This approach in [9] has several limitations such as dynamic adaptation.

Schema is presented based on context observation as a solution of adapting component-based systems during execution time [10]. The proposal focuses on architectural models.

III. Problem and the Proposed Solution

Previous studies tried to solve mismatch problem of components (mismatch between required/expected interface and provided interface of reusable.

This paper proposes a solution to solve adaptation problem catering performance issues of system. The proposed solution just contains explanation of return value mismatch other than remaining mismatch (operation name, method of operation and parameter type which are supported at previous solutions).

3.1 Save System Performance

Firstly, the proposed solution focuses on performance of system that is wasted during component integration and adaptation process. The performance issue is some time raised due to number of extra functions that are provided by component. The number of provided functions may be more than required functions. Loading time of unnecessary functions into a system consumes system resources such as memory and CPU. The 'Functions Detector' tool is proposed to save the system performance. This tool checks on component implementation. The tool contains a database to store information of required operations for current application (adaptation specification) and existing component specification (component operations name, parameter type and method of operation).

This tool does this:

The proposed tool will select each existing/provided function and check with the required function.

- If this function is needed and required in the new system it will be adapted to match required function interface?
- If this function is not needed then the operation/function code will be omitted from existing component?

This step will reduce loading of irrelevant functions to save time. After this process, the component will be integrated into system and the adaptation of interface component will be provided. Figure1 shows the working of 'Function Detector' tool.

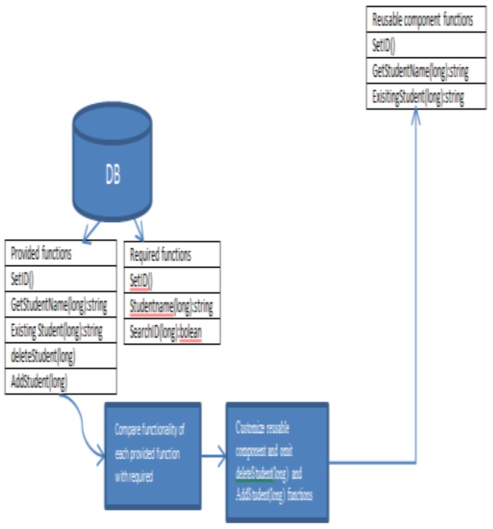
Reusable component functions
SetID()
GetStudentName(long):string
ExisitingStudent(long):string
DB
Provided functions
SetID()
GetStudentName(long):string
Existing Student(long):string
deleteStudent(long)
AddStudent(long)
Required functions
SetID()
Studentname(long):string
SearchID(long):bolean
Compare functionality of each provided function with required
Customize reusable component and omit deleteStudent(long) and AddStudent(long) functions

Fig 1: Function Detector Tool

Table 1: Comparison of brief literature review

Paper Title	Limitations
Research on Behavior Adaptation of Software Component	If there is any missing data the deadlock will happen and the message will lose from system and calculate the synchronous relations among interacted components and reduce it
Research on Safe Behavior Adaptation of Software Component	Time consuming and need to remember the order of message and make them reorder.
Safety Verification of Software Component Behavior Adaptation	Time consuming and need to remember the order of message and make them reorder.
Component Adaptation Mechanism	platform dependent adaptation system based on purely BCA methodologies understand each static information of the component structure and the classes
Automatic Synthesis and Adaption of Gray-box Components for Embedded Systems Reuse vs. Optimization	non-functional properties for instance are not taken
Validation of Novel Approach to Detect Type Mismatch Problem Using Component Based Development	Reduce performance of system
Automated Creation and Assessment of Component Adapters with Test Cases	Depend on a brute-force (all possibilities) that leads to consuming of time and resource. Also it slow the response for user research query
Leveraging Software Search and Reuse with Automated Software Adaptation	The current adaptation algorithm only considers methods to be compatible if their signatures Fully match.
Unifying Design and Runtime Software Adaptation Using Aspect Models	Dynamic adaptation of dynamic adaptations Weaving order at runtime. Paradigmatic dependence.
An MDE Approach for Runtime Monitoring and Adapting Component Based Systems: Application to WIMP User Interface Architectures	Computational cost due to M2M transformation consume the system performance

3.2 Component Interface Definition Mismatch Solution

The problem is not mismatch between functionality of the required functions and the reusable component implementation. This is due to a syntactic mismatch of the interface definition. The study in [5] proposed a solution to detect mismatch between required and provided interface but it didn't solve fully semantic mismatch. This paper proposed a solution to add a check on return value mismatch. First check for each mismatch between required interface component and interface of existing component in fully way such as mismatch of operation name, parameter type, method of operation and mismatch of return value.

Adapted component (adapter) is to match the provided functions with required with fully semantic match. Adapted component contains modified operation name, methods of operation and return value this component may contain little of code. The proposed solution cares about fully semantic match between adapted and required interface. Adapted component acts as translator between the system and existing component which is understand the request from client and understand reusable component interface. It receives request from clients and send it to existing component then it sends the response again from existing component to client. Table 2 shows examples of match and mismatch between required and provided functions.

Table 2: Required and provided interface example

Required functions	Provided functions	match
SetID()	SetID()	yes
StudentName(long): string	GetStudentName (long): string	no
SearchID(long): Boolean	SearchID(long): String	no

As shown in Table 2, there is an example showing the return value mismatch in function SearchID (long). The function has name and parameter type match but there is no return value match. The required return value of SearchID (long) is Boolean while the provided function has String as return value of same function as shown in figures 2 and 3.

Required function:

```
SearchID (long ID)
{
if there is student
Return true;
Else
Return false
}
```

Fig 2: Required SearchID()

Provided function:

```
SearchID (long ID)
{
String st1, st2;
St1="there is student";
St2="there is no student";
If there is student
Return st1;
Else
Return st2;
}
```

Fig 3: Provided SearchID()

IV. Validation

Survey is used as a research design to validate the proposed solution. It consists of 20 questions belonging to 4 goals. Each goal has 5 questions. This survey is distributed online among different developers and designers who are interested in component based development (CBD) and component adaptation.

Likert scale ranges from 1 to 5 as follows.

- Strongly Disagree indicating 1
- Disagree indicating 2
- Neutral indicating 3
- Agree indicating 4
- Strongly agree indicating 5

Thirty respondents fill the data and statistical analysis is performed using frequency tables and bar charts.

4.1 Cumulative analysis of first goal

Goal 1: Considering the important of performance system during component adaptation**.**

The results of survey of this goal are shown in Figure 4. It clears from the cumulative descriptive analysis of goal 1 that 49.33% of the sample agreed to take care of system performance during component adaptation and 24.67% strongly agreed while 20.67% are neither agreed nor disagreed whereas 4.67% are disagreed.

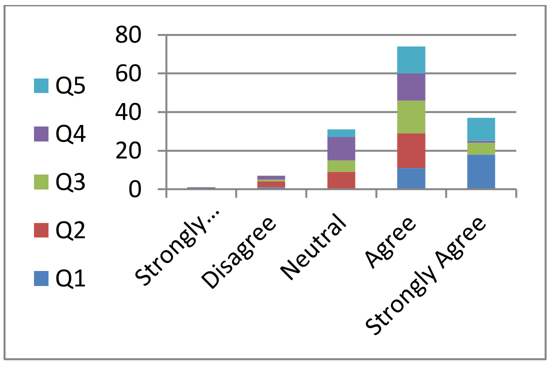

Fig 4: Graphical representation of cumulative statistical analysis of goal 1

4.2 Cumulative analysis of second goal

Goal 2: Consider Considering fully semantic match through component adaptation.

Figure 5 shows the cumulative descriptive analysis of goal 2 that 46% of the respondents are agreed and 26% are strongly agreed with importance of fully semantic during component adaptation while 10% are neither agreed nor disagreed whereas 17.33% are disagreed.

Table 3: Cumulative Statistical Analysis of Four Goals

Q. number	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1	0	1	0	11	18
2	0	3	9	18	0
3	0	1	6	17	6
4	1	2	12	14	1
5	0	0	4	14	12
6	0	0	2	11	17
7	1	24	5	0	0
8	0	0	3	20	7
9	0	1	3	22	4
10	0	1	2	16	11
11	0	0	5	9	16
12	0	2	20	8	0
13	1	5	16	8	0
14	0	0	5	9	16
15	0	1	5	20	4
16	0	9	18	3	0
17	0	10	16	4	0
18	0	1	19	8	2
19	0	0	5	8	17
20	0	0	3	8	19
Total	3	61	158	228	150
Average	0.50%	10.17%	26.33%	38.00%	25.00%

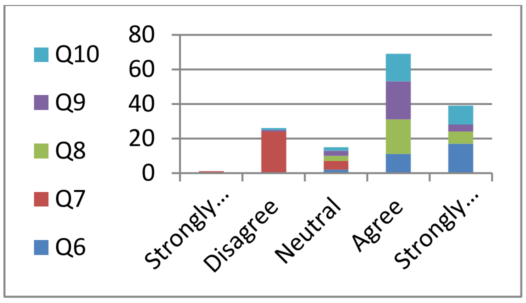

Fig 5: Graphical representation of goal 2

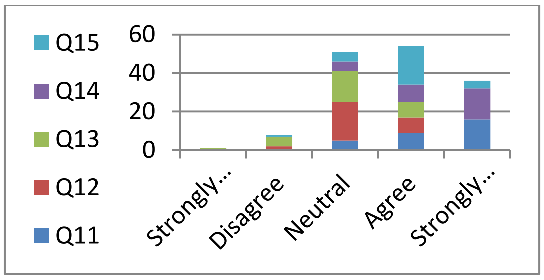

Fig 6: Graphical representation of goal 3

4.3 Cumulative Analysis of third goal

Goal 3: Considering return value through representing component interface.

The results of survey for goal 3 are shown in figure 6. It displays that 36 of the sample agreed and 24% are strongly agreed with importance of considering return value through representing component interface while 34% are neither agreed nor disagreed whereas 5.33% are disagreed.

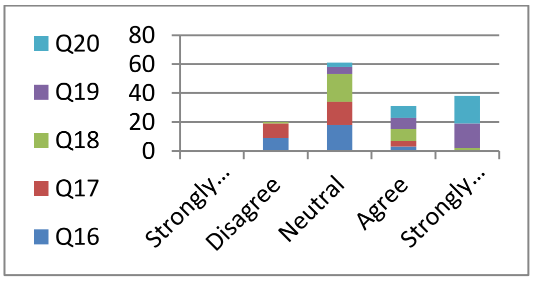

Fig 7: Representation of cumulative statistical analysis of goal 4

4.4 Cumulative analysis of forth goal

Goal 4: Detecting return value mismatch.

The survey's results of goal 4 are shown in figure 7. It depicts that 25.33% of the respondents are strongly agreed and 20.67% are agreed while 40.67% are neither agreed nor disagreed whereas 13.33% are disagreed.

4.5. Comprehensive Cumulative analysis of four goals

Table 3 shows that 38% of the respondents are agreed and 25% are strongly agreed to all of the four goals. 26.33% are neutral while 10.17% are disagreed and 0.50% strongly disagreed.

V. CONCLUSION

Software component adaptation has become a crucial problem in component-based software engineering (CBSE) as a result of mismatch and incompatibility between the interfaces for the purchased components and the required functions during assembling and reuse. This paper proposed a solution to solve this problem by focusing on detecting fully semantic mismatch without sacrificing system performance. The solution is to include return value mismatch next to operation name, method of operation and parameter type mismatch. A 'Detector' tool is proposed to assigns and deletes unwanted functions from reusable component to save system performance before integration process. The proposed solution is validated using a survey. It is concluded from the cumulative analysis that system performance is an important factor during adaptation of components. Return value mismatch has high effect during component adaptation process. Therefore, return value mismatch should be taken into consideration while selecting component interfaces using the proposed 'Detector' tool.

REFERENCES

[1] Sae Hoon Kim; Jeong-Ah Kim, "Component Adaptation Mechanism," Ubiquitous Computing and Multimedia Applications (UCMA), 2011 International.

[2] Xiong Xie; Weishi Zhang; Xiuguo Zhang; Zhiying Cao; Jinyu Shi, "Research on Safe Behavior Adaptation of Software Component," Computational Intelligence and Software Engineering (CiSE), 2010 International Conference, pp.1-4, 10-12 Dec. 2010.

[3] Xiong Xie; Weishi Zhang; Xiuguo Zhang; Zhiying Cao; Jinyu Shi, "Research on Safe Behavior Adaptation of Software Component," Computational Intelligence and Software Engineering (CiSE), 2010 International Conference, pp.1,4, 10-12 Dec. 2010.

[4] Xiong Xie; Weishi Zhang; Zhiying Cao; Xiuguo Zhang; Jinyu Shi, "Safety Verification of Software Component Behavior Adaptation," E-Product E-Service and E-Entertainment (ICEEE), 2010 International Conference, pp.1,4, 7-9 Nov. 2010.

[5] Borde, E.; Carlson, J., "Automatic Synthesis and Adaption of Gray-Box Components for Embedded Systems - Reuse vs. Optimization," 35th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2011, pp.224-229, July 2011.

[6] Rizwan Jameel; Ebtesam Alomari," Validation of Novel Approach to Detect Type Mismatch Problem Using Component Based Development," information technology and computer sciences, pp. 108-117 August 2013.

[7] Hummel, O. and Atkinson, C.: Aut omated Creation and Assessment of Component Adapters with Test Cases, International Conference on Component-Based Software Engineering (CBSE), Prague, 2010.

[8] Janjic, W.; Atkinson, C., "Leveraging software search and reuse with automated software adaptation," Search-Driven Development - Users, Infrastructure, Tools and Evaluation (SUITE), 2012 ICSE Workshop, pp.23,26, 5-5 June 2012.

[9] C. Parra, X. Blanc, A. Cleve, and L. Duchien, "Unifying design and runtime software adaptation using aspect models," Science of Computer Programming, 2011.

[10] Criado, J.; Iribarne, L.; Padilla, N.; Troya, J.; Vallecillo, A., "An MDE Approach for Runtime Monitoring and Adapting Component-Based Systems: Application to WIMP User Interface Architectures," Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference, pp.150,157, 5-8 Sept. 2012.

Authors' Profiles

Aisha Mohammed Alshiky is a master student in IT Department at King Abdulaziz University, interested in CBD and Technology Management.

Dr. M. Rizwan Jameel Qureshi received his Ph.D degree from National College of Business Administration & Economics, Pakistan 2009. This author is best researcher awardees from Department of Information Technology, King Abdulaziz University Saudi Arabia in 2013 and Department of Computer Science, COMSATS Institute of Information Technology Pakistan in 2008.

How to cite this paper: Aisha Mohammed Alshiky, M. Rizwan Jameel Qureshi,"Detecting Return Value Mismatch during Component Adaptation with Concern of System Performance", IJIEEB, vol.6, no.2, pp.41-46, 2014. DOI: 10.5815/ijieeb.2014.02.05