A Generative and Component based Approach to Reuse in Database Applications

Beihu Wang [1], Xiaodong Liu [1], Jon Kerridge [1]

[1] School of Computing, Napier University, Edinburgh, Scotland UK
{t.wang, x.liu and j.kerridge}@napier.ac.uk

Abstract. The development of database application systems will benefit from high reusability because similar design circumstances recur frequently in database developments. However, research in software reuse has shown that mismatches of components with the application architecture, state and other components, destroy the component reusability. In this paper, a generative and component based reuse framework is presented to tackle the problem of high variability and therefore to achieve higher reusability in database application development. A Scenario based dynamic component Adaptation and GenerAtion technology (SAGA) is developed to support deep component adaptation and component generation. XML has been used as the universal information carrier in the approach.

1 Introduction

Software reuse has shown very successful improvement on efficient, high quality and low cost software design and development. However, reuse in database domain seems not to have received enough attention. Until now the most of the reuse research has been done only on the reuse of general software systems [9][10]. Approaches and tools for improving reusability in database application development are urgently needed.

Research in software reuse has shown that "as-is" reuse occurs extremely little and that components generally need to be changed in some way to match the application architecture, state and other components. Based on the existing component adaptation technologies, a component can be altered in several ways. As the adaptee is a black-box component, the alteration is restricted to what can be done using its public sets, such as conversion of parameters, modification of access mechanism, and extension of functionality. And as inheritance and wrapping techniques are popularly used, adapted component can be too complex and less reusable. Therefore, the problem of component mismatch has not been well solved by existing component adaptation techniques, due to the lack of component information, high redundancy, and shallow adaptation.

In this paper, a generative and component based reuse framework is introduced to achieve high reusability in data-intensive system development. SAGA is developed in the framework as the core technique. It is often the case that provisionally qualified components still have some architectural and behavioural inconsistency with the requirements of a specific application system. To eliminate these inconsistencies, adequate adaptation technology must be applied to the components. For adaptation in very depth, the new component instance may need to be created with generation technology. In this PhD project, we propose to use a set of scenarios to record the design configuration of components reused in specific applications. Scenarios may be adjusted, composed or associated interactively to cope with complex reuse cases. As the foundation, a comprehensive component specification is required for component's understanding, use and implementation. As the XML documents are easy to understand by both human and machine, an XML formatted Component Definition Language (CDL) is designed to represent component specification.

2 Existing techniques

Some related works have been done to improve the reusability of components. These works can be summarised into the following three categories.

2.1 Component Customisation

Component-users customise a software component by choosing from a fixed set of options that are already pre-packaged inside the software component. A component should be customisable during the reuse to fit itself into specific real-world requirements. Based on this perspective, component customisation [12] is popularly used in component-based development. Because the modification is limited on certain predefined options, it may not be possible to satisfy the additional requirements.

2.2 Component Adaptation

Component-users adapt a software component for a new use by writing new code to alter existing functionality. If an application builder decides to adapt the component, the application builder must understand the complex behaviour and functionality of its classes, so any change in either of them will not break the structure of the system specified by the component designer. However, in most adaptation mechanisms [1][7][8], such as inheritance, wrapping, active interfaces, binary component adaptation, and superimposition, the adaptee is black-box components. So without enough understanding about component, the modification is limited on some shallow level, such as conversion of parameters, refinement of operations, modification of access mechanism, and extension of functionality. And also, with the more layer code addition on existing code, the redundancy of component could affect system efficiency.

2.3 Component Generation

Component-users generate a software component by accepting a configuration description to be interpreted by their configuration code and assemble the concrete components according to this description. It is not sufficient just to be able to write configuration code. It is also needed to deploy some design principles to organize the code in a clean way. One such principle is to encapsulate configuration knowledge, and then based on the configuration to generate components. In existing component generation systems [2][3][6][11], configuration is composed by fixed options, provided by component-developers. Component-user can only modify components in limited choices, which may not be possible to satisfy the additional requirements.

3 The Framework

It was advocated that the initial start point of component reuse is requirement analysis. In this stage both data and processing requirements will be decomposed and represented with data-intensive use cases, which is an extension of UML use cases with database application features and rigorous descriptions. The selection of suitable components happens at the architectural design stage. Based on the requirements expressed in data-intensive use cases, the architectures of the data schema and processing software will be developed. Component qualification will be done in parallel with architectural design, which means that the selection of components should comply with the architecture of the database application. Pre-qualified components are selected from the component repository constructed during component mining. Components repository involves Component Specification (CS) presents in CDL, and primitive component code, which is used to compose the selected component.

During component adaptation, design configurations will be collected interactively with the component-user. These design configurations define how the selected components can be adapted from existing primitive components for use in a particular database application. The design configurations are presented as scenarios, which consist of a serial of component adaptation actions to satisfy particular adaptation requirements. According to the scenario, component adaptation process will generate adapted Component Derivative Instance Specification (CDIS). Final products of architectural design are qualified CDIS and the architectural model, which validates the requirements.

The next stage is component generation and integration. The qualified component, which may be parts of the database schema or processing transactions or both, will be generated, based on its CDIS. The process is cyclic, i.e., the design configurations may be adapted or refined further to fit better to the particular database application, and then more suitable components may be generated or adapted until the software engineer is satisfied so the components can be integrated into the application.

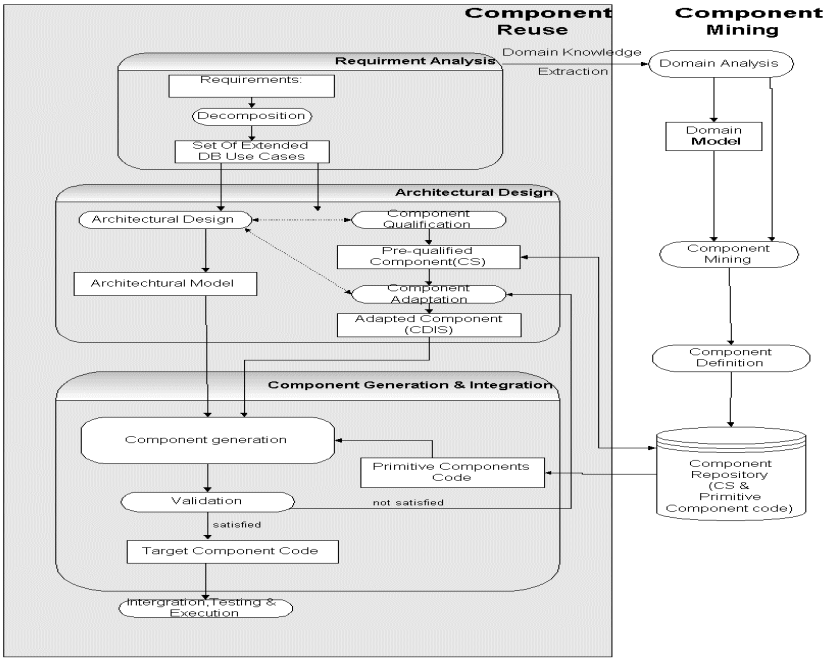

Fig. 1. The Generative Component-Based Reuse Framework

Finally, the generated components (implementations) are integrated in the system integration stage. For database applications, the generated components may contain a data schema part and a transaction part, or both.

4 Scenario based dynamic component Adaptation and GenerAtion technology (SAGA)

The rationale underlying scenario based deep component adaptation is that pre-qualified components must be adapted to eliminate the conflicts to specific reuse requirements and these conflicts can be eliminated with adequate adaptation and generation based on correct reuse requirements captured in scenarios. The pre-qualified component is defined in CS. CS defines the overall capability of the component. It involves signature, constraints and non-functional properties.

Based on the above observation, we have identified that component-based development, in addition to a set of reusable components, requires a set of scenarios. A scenario is composed of a series of adaptation actions, which are defined in adaptation types. With scenario and generation, deep component adaptation becomes feasible.

Scenarios may be required in three aspects: 1) the component may be used in different reuse contexts; 2) the component may be used under different constraints; 3) the

component may act different roles in given reuse contexts. Scenarios can be composed, associated and adjusted to tackle various and complex reuse cases. The role of scenarios can be defined as the design configuration of components in specific kinds of applications. The pre-qualified component is defined in a CS. Scenarios aim to perform the adaptation actions on CS instead of component code. The result will be the specification of the adapted component derivative, namely CDIS.

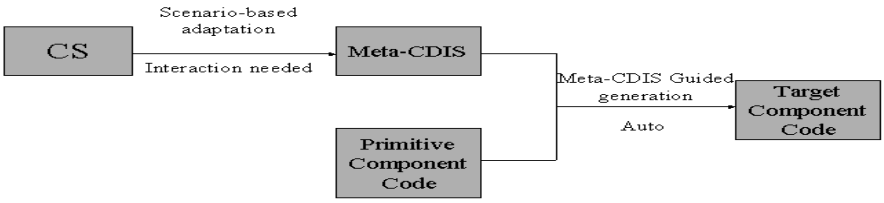

Fig. 2. The process of SAGA

Adaptation actions are grouped into relevant component adaptation types. An adaptation action is defined with the following attributes and sub-statements: (1) *name*, which is the identity of the action; (2) *type*, which indicates the type the action; (3) *requirement*, which shows the reuse context the adaptation action suits for;(4) *position*, which shows a position in CS where the actions to be done; (5) *extra detail*, which covers other details needed by the action. A typical scenario of component adaptation is given below.

```
<scenario ID=" xxxxx " >
    <adapt_action  name="xxxxx " type="xx_xxx">
      <requirement>xxxxxx</requirement>
      <position>xxx.xxx.xx</position>
       ……
       extra detail
       ……
    </ adapt_action>
    <adapt_action  name="xxxxx " type="xx_xxx">
    ……
    </ adapt_action>
</scenario>
```

5 Conclusion

Based on the observation that similar design circumstances recur frequently in the development of database applications, we concluded that raising reusability in database applications would improve the efficiency of development greatly. The fact that most reuse approaches and tools have been concentrated on general software systems with little emphasis on reuse in a data intensive environment has triggered the research in this paper.

As a reuse methodology, the current framework aims to facilitate the database application development with improved reusability. Components in our approach aim to be highly adjustable or generateable based on the design configurations and primitive components. These components are the reusable blocks to build a new database system. Existing expertise and idioms of database design recorded in these components are then presented as the starting point for the design of a new database application. Components are described in the XML-based CDL and stored in the component repository. The component definition language is a semi-semantic definition language. The features inherited from XML make the components easier to understand, to exchange and to propagate over software communities.
The SAGA technology gives a great potential for coping with component incompatibilities, it meanwhile reduces component overhead. A scenario gives component-users understandable and interactive component adaptation information, and components generated based on the scenarios will enjoy high suitability and efficiency in particular applications. As a PhD project, the work presented in this paper is ongoing. Our initial case studies have shown the work is promising.

References

1. Bosch, J., Superimposition: A component adaptation technique, Technical Report TR, Department of Computer Science and Business Administration, University of Karlskrona/Ronneby, (1997).
2. Batory, D., Composition Validation and Subjectivity in GenVoca Generators, Software Engineering, (1997).
3. Biggerstaff, T.J., A characterization of generator and component reuse technologies, Generative and Component-Based Software Engineering, (2001).
4. C.J.EGYHAZY, From software reuse to database reuse, in International journal of software engineering and knowledge engineering, 10: pp. 227-249, (1998)
5. Han, J., A comprehensive interface definition framework for software components,Asia-Pacific Software Engineering Conference, Taipei, (1998).
6. Hans de Bruin, H.v.V., The future of component-based development is generation, not retrieval, 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems
7. Geoge T. Heineman, H.M.O., An evaluation of component adaptation techniques, Carnegie Mellon Software Engineering Institute, (1999).
8. Ralph keller, U.H., Binary Component Adaptation, Department of Computing Sciense, University of California, (1997).
9. TATJANA WELZER, M.D., Similarity Search in Database Reusability -- a Support on Efficient Design of Conceptual Models., *Vaasa: Vaasan yilopisto,* (2000).
10. TATJANA WELZER*, B.S., Reuse Database Components, the Patterns From MetaBase Repository , *HAWAIIAMS98,* (1998).
11. Ulrich Breymann, K.C., Generative components: one step beyond generic programming, Generic Programming, (1998).
12. *Stephen S. Yau, Choksing Taweponsomkiat,* An Approach to Object-Oriented Component Customization for Real-time Software Development**,** Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing**,** (2002).