

## An ADL for Component Software Development

Student: Issac Laplace

Faculty Director: Andy Ju An Wang

Thesis Project Time: Spring 2003 – Fall 2003

1. **Subject Areas:** Software engineering, Component-based development, Embedded systems, Architectural design, and Compiler construction
2. **Principal goals of the project:** Design and implement an ADL (architectural design language) for component-based software development. Starting from the DBSL (device bean specification language), this project will build a compiler accepting DBSL programs as input and generating XML-based architectural description as output. The XML file could be loaded into a component tool like Bean Builder 1.0. One of the benefits of such an ADL is to provide a formal basis for component-based development and thus software developers can formally reason the system at the architecture level and checking consistency and completeness of a component-based system.

Moreover, this project will build an IDE (integrated development environment) for component-based software development using the ADL and its compiler developed in this project.

Component is the key concept in CBSE (component-based software engineering) and software architecture. CBSE focuses on the realization of systems through integration of pre-existing components. CBSE is motivated by the decreasing lifetime of applications and the need for more rapid product development. Designing reusable components for embedded systems is more complex than designing components in traditional software components. This is due to the characteristics of embedded systems, limited memory, processing power, heavy workload, shorter-time to market, to name just a few. CBSE provides a very promising way to build embedded software.

Device beans refer to the reusable components for embedded systems. Device beans represent a component model, capturing what is a reusable component and how it will behave. Some existing component models include COM/DCOM, CORBA, JavaBeans, OSGi bundles, and .NET assembly. Comparing to these existing component models, device beans is designed towards embedded systems and embedded applications. Device beans encompass many good features including interoperability, portability, and visual support through a device bean design environment. Embedded software consist of largely various device drivers making up an application. As embedded computing becomes ubiquitous, it is important to develop embedded software using the start-of-art software development approaches such as aspect-oriented programming and component-based development.

Many modeling tools and specification languages are available for embedded systems: UML, SpecC, SystemC, SDL, Rhapsody, to mention just a few. As concerns of design, however, few tools are ready in use for embedded systems. This is especially true for design tools supporting specific design method or strategy. For instance, by the time of writing this proposal, we were not aware of any design tools or environment supporting component-based design and development of embedded systems.

Component-based design has two fundamental activities: design *for* reuse, and design *with* reuse. While the major purpose of design for reuse is to establish a comprehensive reusable component library, the major purpose of design with reuse is to build new products by reusing pre-built components. As component-based development method becomes popular and promising in

building conventional systems, it is interesting to see how well it can be applied to embedded system area. Compared with conventional systems, the distinguish features of an embedded systems include its inherent complexity and high dependency of interactions between hardware and software. Embedded systems are typical hybrid systems, because they often have to deal with discrete time and continuous time, analog signals and digital signals within a single system. Recent research work like [Lisboa] and [Kuhn] showed that object-oriented approach could be applied to embedded systems. Our purpose on CBDK is to move a step further: applying component-based approach to the development of embedded systems.

CBDK provides basic functionality for designers to build embedded systems with component-based approach. Although CBDK also supports source code editor and code generation, it is much simpler than a full-blown IDE like Forte for Java, JBuilder, or VisualAge. However, it provides more functionalities than BDK and Processor Expert.

BDK [Sun Mic] was developed by Sun Microsystems as a visual design environment for Java beans. It does not have a source code editor, neither a compiler nor a JAR utility, thus BDK is not good for an IDE to carry out the first task in component-based development mentioned previously: design for reuse. The user has to use another IDE to generate a Java bean packaged in a JAR file, and load it into the "ToolBox" of BDK. Besides, BDK does not provide any method editor or event editor. Therefore, the use of BDK cannot modified a Java bean other than simply change some public properties. Furthermore, BDK provides only two operators to integrate components: plus and multiplication. With our CBDK, however, a user can generate a new component, called device bean, by either manually coding in a source code editor, or by modifying any items in properties, methods, or events. Moreover, CBDK has a comprehensive integration environment, providing more assembling operators for the user to perform the task of design with reuse.

Processor Expert [UNIS] provides a rapid development environment for embedded applications. A software prototype can be constructed based on hardware components, called *embedded beans*. However, it does not provide ways of analysis, source code editing. Also it does not provide various operators for component integration. Furthermore, only a limited number of microprocessors are supported in Processor Expert. Our CBDK is not only a prototyping tool but also an integrated environment supporting component-based development of embedded systems. It supports both design for reuse and design with reuse. It is implemented in Java to be able to run on any platform with Java virtual machines.

MBeans or management beans [Sun Mic] are reusable components used in JMX (Java Management Extension) targeting enterprise computing and enterprise management. However, our device beans are targeting embedded systems and smart networked appliances.

3. **Resources Required:** An operating system (any of Linux, Windows NT, Windows 2000, or Solaris); JavaCC, JDK 1.4.x; BDK 1.1, and Bean Builder 1.0. Contact the supervisor for further details.
4. **Degree of Difficulty (addressed by the faculty advisor):** Easy for students with compiler writing experience, especially with JavaCC experience. Moderate for students with experienced Java programming skills. Challenging for students without Java programming experience.
5. **Background Needed (addressed by the faculty advisor):** The student must be fluent in Java and familiar with event-driven programming. Knowledge about JavaCC, computer architecture, compiler construction will be of help. Contact the supervisor for further details.
6. **References:** Component-based development, JavaBeans, Device beans; JavaCC, ADL, Embedded systems and programming, and IDE development.
7. **Tentative Working Schedule and Deliverables:**

| <i>Time</i> | <i>Tasks</i>   | <i>Deliverables</i>                           |
|-------------|--|---|
| 1/21 – 1/31 | Form an evaluation committee, Getting familiar with Bean Builder 1.0 | Formal proposal<br>5 examples of ADL programs |
| 2/1 – 3/31  | Complete ADL syntax and semantics with adequate case studies         | Syntax of DBSL<br>Semantics of DBSL           |
| 4/1 – 5/31  | Learn JavaCC   | A prototype of the compiler from DBSL to XML  |
| 6/1 – 8/31  | Building the compiler  | Compiler version 0.5                          |
| 9/1 – 10/31 | Testing  | Testing and debugging<br>Compiler version 1.0 |
| 11/1 – 12/5 | Integration and presentation   | Thesis  |

## 8. **Outline of the Thesis:**

### Chapter 1 Introduction

- 1.1 Problem statement
- 1.2 Research challenges:
  - 1.2.1 Why this research is important?
  - 1.2.2 Why this research is difficult?
- 1.3 Related work
- 1.4 Major contributions and achievement of this research
- 1.5 Thesis organization

### Chapter 2 Related Work and Existing Solutions

- 2.1 Research background
- 2.2 ADL research work
- 2.3 Wright, ACME, and C2
- 2.4 Other ADLs
- 2.5 Implementation technology
- 2.6 IDE for COP
- 2.7 Comparison and conclusion

### Chapter 3 DBSL Syntax

- 3.1 From device drivers to embedded software
- 3.2 Specification of DBSL
- 3.3 Examples of DBSL specifications
- 3.4 Component-based development with DBSL

### Chapter 4 DBSL Semantics

- 4.1 Component charts
- 4.2 Static structures
- 4.3 Dynamic behaviors
- 4.4 Component traces

### Chapter 5 Compiler Implementation

- 5.1 Overview of the compiler implementation
- 5.2 YACC approach
- 5.3 JavaCC approach
- 5.4 Our approach to implementing DBSL compiler
- 5.5 Software architecture
- 5.6 Component-level design
- 5.7 Implementation details
- 5.8 Summary

## Chapter 6 The IDE Design

- 6.1 IDE overview
- 6.2 NetBeans approach
- 6.3 YES-CBDK
- 6.4 A centralized reusable component base
- 6.5 User interface
- 6.6 Typical usage scenarios
- 6.7 Further improvement

## Chapter 7 Summary and Discussion

- 7.1 Major innovative contributions of this research
- 7.2 Related work
- 7.3 Future work

## References

[Sun Mic] Sun Microsystems, Inc., <http://www.sun.com> November 2002.

List other references following the reference guide of IEEE style at <http://lovelace.spsu.edu/jwang/student/ReferenceIEEEstyle.html>.