COMPONENT-ORIENTED SOFTWARE DEVELOPMENT IN JAVA

THESIS

Submitted to the Graduate Council of
Texas State University-San Marcos
in Partial Fulfillment of
the Requirements

For the Degree

Master of SCIENCE

By

Micah Spears, B.S.

San Marcos, Texas
December 2003

ACKNOWLEDGEMENTS

I would like to thank Dr. Greg Hall for his advice and guidance in composing this thesis.

I also give thanks to my parents for their continued support of my academic endeavors.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

1.1 Introduction

Software development is the process by which a set of requirements is transformed into a verifiable software system. The rationale for the process is represented by the logic employed to design and implement the various design abstractions. Within these abstractions lies the root complexity of software development itself, the programmable logic. Software Developers tend to cope with complexity by decomposing the implicit convolution into smaller more intelligible modules or components Frakes and Pole (1994). This narrowing of scope allows the person to focus on a subset of logic represented by the system.

Object-Oriented (OO) software development is the process by which intangible software requirements are transformed into tangible objects Frakes et al. (1994). Component-Oriented (CO) software development is the concept that extends the notions of OO development by incorporating a high-level abstraction layer. The component abstraction layer serves as a surrogate view of the software system to help simplify the overall system architecture. By grouping and isolating logical components, the modular system can provide feature and behavioral isolation to help prevent and isolates software faults.

This thesis examines how Component-Oriented software development is used to simplify the overall complexity of a given OO software system by drawing upon proven software engineering principles such as the divide-and-conquer technique. The divide-and-conquer technique is the process by which a complex problem is decomposed into smaller, workable units thus simplifying the problem. The OO programming language subject to natural componentization is Java due to its inherent Contract-Based (CB) programming level constructs that can provide static and runtime software quality assurance. The CB constructs consist of a formal software interface for type checking and assertion facilities for validation and verification. Software systems are privy to frequent requirements changes and they must be built or modified on challenging development schedules (Liu). These constructs are employed to develop and implement a component language extension for the Java Runtime Environment (JRE) that provides compile-time dependency checking, runtime isolation, and contractual interface binding. To verify the benefits of a language level CO process, a quantitative software analysis is performed comparing the CO facilities to the standardized OO facilities at the design and language level.

1.2 Elaboration of Thesis Statement

OO software development has been proven to simplify complex software systems through layered abstraction and information hiding to provide reuse. Software reuse is an important area of software engineering research that promises significant improvements in software productivity and quality Frakes et al. (1994). Abstraction is a technique used to explain to a client what must be revealed so he/she knows how to use a module; i.e.,

the mathematical model needed to reason about client programs (Harms, 1991). The abstraction layers represent the dynamic behavior of the system by wrapping the respective behavior in a façade, an object. This layering alleviates the need for the developer to understand the internal workings of that object hence providing behavioral hiding. Information hiding is the lowest level of abstraction where the behavior of the object governs the manipulation of the underlying data. Information hiding is a technique whereby concrete representations used within a module are kept secret from a client (Harms, 1991). This data abstraction is purposely hidden from black-box observers through object encapsulation.

CO software development is a natural extension to the information and behavioral hiding inherent in objects. This extension must be both natural and conducive to the programming language environment, Java, for developer adoption. Furthermore, the component extension must provide static dependency checking, dynamic runtime isolation, and contract negotiation at the component level. These component attributes help assure the software product is adequately monitored at runtime for potential faults. Current component models fail to provide all of these basic functionalities to the software developer due to runtime limitations or design compromises.

This thesis asserts that a language level component model can provide both static and dynamic capabilities while achieving the theoretical goals of component design. Previous attempts at component models such as Jiazzi are shown to be inadequate with realizing the full benefits of CO techniques McDirmid, Flatt, and Hsieh (2001).

Furthermore, CO design is shown to supercede OO design in a quantitative analysis using proven Software Engineering (SE) metrics.

1.3 Define "Object"

The formal definition of an object is

> "Something perceptible by one or more of the senses, especially by vision or touch; a material thing." (Merriam-Webster, 1989)

In software terms, an object is an abstraction of computational logic whose purpose is to simplify the software concept at hand. The object represents a concrete entity modeled in a real world context thus allowing humans to visualize the software logic in a natural way.

Once the object is abstracted, a class is fashioned to realize the purpose of the object. The class will contain methods that operate on the data of the class much like a real world object has functions that exist to perform tasks.

Objects are realized during runtime by way of instantiation from an instance of the respective class. In contrast to components, an object is a unit of instantiation; has state, and encapsulates its state and behavior Kiziltan, Jonsson, and Hnich (2000). Thus, instantiation is analogous to the birth of a biological entity. Similarly, the death of a biological entity coincides with the finalization or deconstruction of a software object.

1.4 Define "Component"

The formal definition of a component is

> "A fundamental, essential, or irreducible constituent of a composite entity within a system." (Merriam-Webster, 1989)

In software terms, a component is an independent collection of co-operating objects with a clearly defined boundary to other objects or components and has no persistent state Kiziltan et al. (2000). In the system architecture, components are the heterogeneous logic of a system is partitioned into unique logical groups i.e. components Kiziltan et al. (2000). Each component symbolizes a section of homogenous logic which can exist independent of other components.

For a component to be a logical group of related logic, it must contain all relevant pieces of the system. The relevant pieces are dependencies and core logic necessary to form an independent unit of deployment within the runtime environment. A component should be able to plug and play with other components so that it can be composed at run-time without re-compilation Kiziltan et al. (2000). For efficiency, the dependences can be shared amongst the divergent components in the runtime environment.

1.5 Component-Oriented Software Benefits

Component-Oriented software development can provide numerous benefits throughout the software design lifecycle. The known benefits include Lui and Cunninghan (2002)

1.) Independency
2.) Adaptability
3.) Structural Abstraction
4.) Behavioral Hiding
5.) Contractual Binding

Independency is based on the hardware component concept which can be deployed independently and is subject to composition by the third parties (Liu). Adaptability is achieved through strong internal encapsulation where communication is exclusive to an exposed interface (Liu). Structural abstraction is the process of collating functionally homogeneous objects into a unified component. Behavioral hiding is the encapsulation of inter-object communication within the component such that the third party is unaware of the details. Contractual binding is the identification of core business operations exposed by the component's interface such that the input and output of said operations conforms to predefined boundaries Lui et al. (2002).

1.6 Component-Oriented Software Risks

Component-Oriented software development is relatively immature when compared to older models such as structured programming. This relative immaturity can pose risks to any software environment where a full understanding of the said model is not available. The risks are illustrated by a lack of component constructs in any modern programming language.

For Component-Oriented development to succeed, the developers must be aware of the current technologies and tools available for component engineering. The key word is engineering because anyone can create a component whereas reusable and adaptable components are engineered. To be reusable, a software component must be: reliable, secure, efficient, and adaptable (Harms, 1991). Reliability is synonymous with the functional testability of the component and represents the greatest risk to component

adoption (Edwards, 1997). Security is an element of the underlying implementation of the component and culpable to future reuse Wohlin and Runeson (1994). Harms demonstrates the efficiency requirement is easily implemented by using a copy and swap routine. Adaptability refers to the component's ability to reshape itself and prove reusable in future domains (Edwards, 1997).

Component constructs are not available in any known programming language. This thesis will attempt to spur the inclusion of such constructs into future languages to realize the benefits of component development. Until language level components are available, developers should choose a component model based upon issues such as: language integration, binary compatibility, and maturity Frakes et al. (1994).

1.7 Existing Component Models

Component models are superfluous in the present technological environment. They vary by vendor, programming language, style, and platform compatibility. The unifying attribute of the differing models is the lack of tight language integration between the component model and component language. This trend exists primarily due to the relative immaturity of the component model and associated development. Numerous industry standard component models exist: Sun Microsystems maintains the Java2 Enterprise Edition (J2EE) component model, the Object Management Group (OMG) maintains Common Object Request Broker Architecture (CORBA), and Microsoft maintains COM (Common Object Model). In addition, the Jiazzi component system exists for Java component development after compilation.

J2EE is a heavy-weight component framework applicable to server-side development. The design style for J2EE is pattern intensive due to the incorporation of best practices. Platform compatibility in J2EE is provided by the Java Runtime Environment (JRE) which can run on multiple operating systems without source code recompilation. While J2EE has numerous strengths including Java language integration, its heavy-weight nature limits its practical uses to small and mid size software projects.

CORBA allows multiple languages to target its component model by defining a meta-language. Leveraging the meta-language, CORBA dictates very little about style to achieve language neutrality. CORBA runs on multiple operating systems but requires porting and recompilation for the target programming language. The lack of mass adoption for CORBA is attributed to its high development complexity because it supports multiple host languages on multiple operating systems.

Microsoft supplies COM for component development in the windows operating system which relies on C/C++ and Java. COM lacks integrated design patterns due to the low level nature of the component framework and ad hoc aggregation reliance. Interoperability is achieved through: aggregation and dynamic interface negotiation Sullivan et al. (1999). Unfortunately, the aggregate and interface negotiation techniques do not interoperate correctly as interface negotiation can not determine the identity of aggregated components Sullivan et al. (1999). It was found that when components were implemented as dynamically linked libraries, linkage problems occurred when services

were changed Sullivan et al. (1999). As such, COM provides few reliable interoperability services to the developer and effectively ties the component to the Microsoft platform.

Jiazzi is a component linker whose purpose is to provide a component linking mechanism for the Java language. The scope of the linking is at the binary byte-code level after the Java classes have been compiled. In essence, Jiazzi components can be thought of as generalizations of Java packages with added support for external linking and separate compilation McDirmid et al. (2001). Since Jaizzi only exists at the binary level, the non-existent component framework and meta-model is a source of great strength and even greater weakness. The strength is no special core language extensions or conventions need to be used in the Java code used to construct a component (). The weaknesses are exposed in their claims McDirmid et al. (2001):

> 1.) Jiazzi can support the addition of features to classes without editing their source code or breaking existing class variant relationships.
>
> 2.) Because sub-classing across component boundaries and cyclic component linking is supported, component boundaries can be placed in the design naturally.

The first claim refutes the idea of employing a component framework to model components at the source code level. Kiziltan (2000) states

> "By the definition of frameworks, a framework can be seen as a circuit board (component framework) where empty positions are waiting for components to take their place. By filling in the empty slots the framework (the circuit board) is instantiated. Requirements are specified telling what the components must conform to in order to be able to do the job in the circuit. In the notion of formal frameworks we have seen that the behavior of the circuit (the framework) can be specified in terms of pre- and post-conditions of the framework, invariants and instantiations, as well as what components are to participate and the static relations among them".

Instead, the component design in Jiazzi is performed after-the-fact whenever all high-level meta-information has been lost to the compilation process. Therefore, their component model should be called a "packager" or "linker" since the standard component attributes: security, adaptability, and independency are not present. The second claim supposes the existing package and class structure has clear boundaries. This assumption is erroneous because most classes have numerous cyclical dependencies upon third party libraries and packages. Thus the linker must make unsupported assumptions regarding the dependency resolution. This could be easily solved if a meta-model was employed to correctly empower the developer with design control at the source code level.

CHAPTER 2

2.1 Software Design Hierarchy

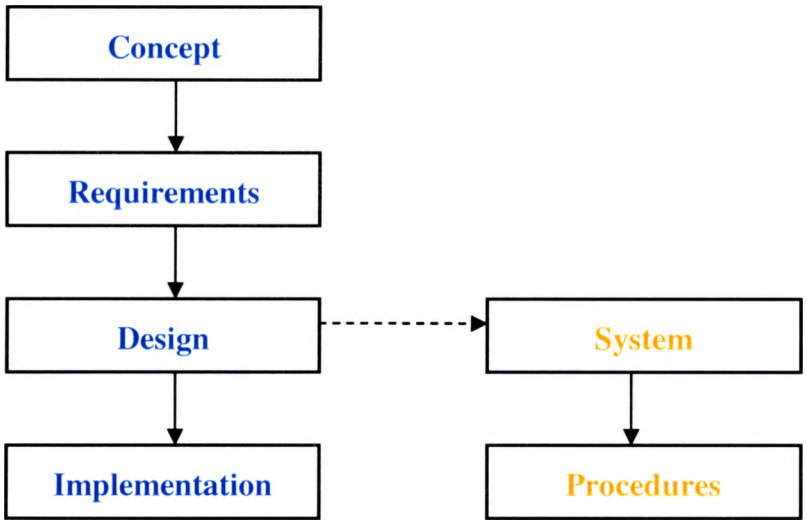
Software Design is the process of transforming business requirements into an abstract, functional model for the system implementation. The model representations are: interface, logical model, and low-level implementation Frakes et al. (1994). Representation methods are important for reuse because they help users understand components and application domains Frakes et al. (1994). Numerous methodologies exist for the requirements-to-model transformation including: ad-hoc, structural, object, component, and a hybrid object/structured design.

Ad-hoc software design is the unstructured, undisciplined approach to designing a software system. Under this philosophy, there are no defined software stages and a software schedule is typically non-existent. Ad-hoc design represents a lack of necessary preparation for the software project and usually results in massive development failure in mid to large systems (Munson, 2003).

In a structured design, the functional elements of the requirements are grouped to guard against unnecessary duplication of common functionalities Khan, Al-A'ali, and Girgus (1995). This method focuses on the inherent functionalities of the system by abstracting the low-level design e.g. procedures. These functionalities are implemented as

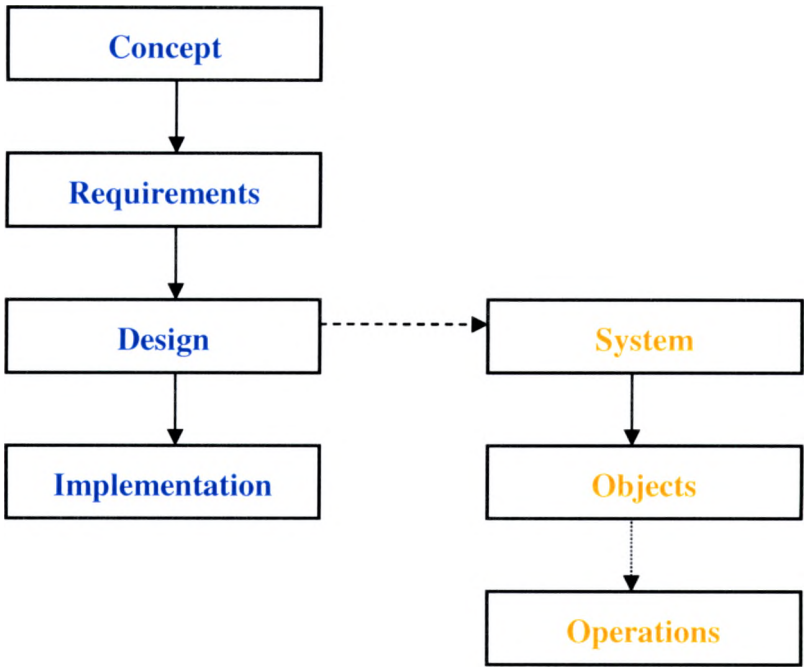
procedures. The procedures call each other to receive, return, or update data values through parameter passing Khan et al. (1995). Figure 2.1.1 illustrates the structured design stages and elements.

Figure 2.1.1



In an Object-Oriented system, the design technique is based upon the transformation of requirements into software objects. Unlike structured development, OO development uses objects, not algorithms, as its fundamental building blocks Khan et al. (1995). To assist in the process, OO designers normally use a technique whereby the nouns from the requirements are extracted and used during the object modeling phase of the design. The compelling reason to use OO design over procedural design is the added level of abstraction; the ability to view the system in terms of tangible objects that encapsulate data. Related benefits include increased reusability, maintainability, and reliability Khan et al. (1995). The object design stages are shown in figure 2.1.2.

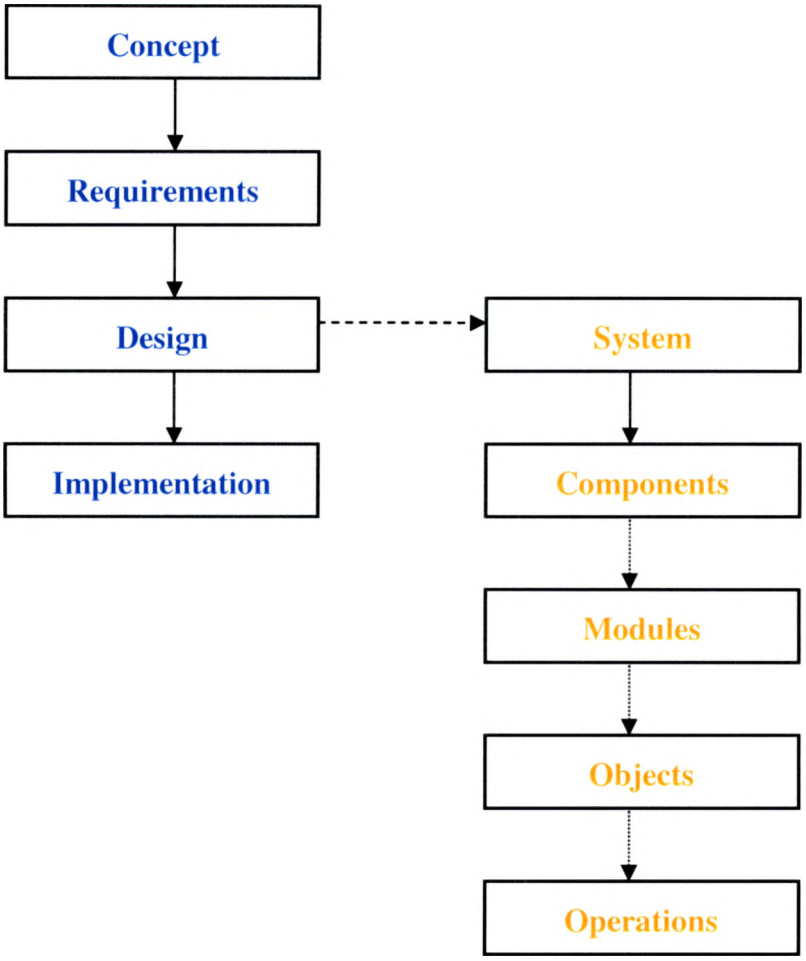
Figure 2.1.2



A key benefit to OO development is the ability to reuse classes in a flexible and intuitive manner through inheritance. Inheritance is implemented via block and white box subclassing where existing classes are extended to override or provide new functionality (Edwards, 1997). Erroneously, white box inheritance is commonly used in software development as the choice method for code reuse (Edwards, 1997). Black box inheritance is a more secure, reliable method for extending the functionalities of an existing class because the data members are effectively hidden from misuse (Edwards, 1997). The OO term for black box inheritance is composition or aggregation. Composition is a "has a" relationship where the enhanced class has a data member, the base class (Edwards, 1997). Aggregation is an enhanced composition relationship that must exist for the normal operation (Edwards, 1997).

Component-Oriented design is the next level of abstraction above an OO design. In CO design, the requirements are logically grouped to form modules. Each module represents a "structured" set of objects analogous to a shared library (Parnas, 1972). A module may also have dependencies upon other modules required at runtime. Components are then built by merging the modules to form an independent unit of deployment. To satisfy the independence requirement, all module dependencies must be included such that no dependencies exist at runtime. Assembled components can then be deployed into an environment like pluggable hardware components. The key benefit of CO design is the inherent independence of the components themselves which results in the high-level decoupling of related software requirements. Figure 2.1.3 illustrates the component design stages.

Figure 2.1.3



As Object-Oriented software systems increase in size and complexity, components are becoming central to the design process, and they deserve close integration with the language McDirmid et al. (2001). At the language level, components can implement (export) one or more interfaces and they can also use (import) interfaces from other components Kiziltan et al. (2000).

2.2 System Design

System design represents the logical workflow of the entire system. The focus is on the interaction between the various entities within the system. The entities within a modern software system can consist of: procedures, objects, and components. A correct system design is achieved by minimizing the coupling amongst the entities and external resource dependencies.

2.3 Component Design

Component design is the process of transforming business requirements into independent, logical entities. Successful component design is marked by first creating a cohesive system design. Without a macro view of the system, requirements grouping cannot be performed with any reasonable degree of accuracy. The requirements grouping is performed by decomposing the requirements into logical, functional groups. If any of the functionalities within the component directly depend upon external resources, the component is not an independent unit of deployment and is thus invalid. The degree of requirements decomposition necessary to provide a reasonable level of abstraction is dependant upon the overall complexity of the system. For instance, a complex system would consist of numerous components and a trivial system would contain only a few components. A component package may consist of Kiziltan et al. (2000):

1) A list of provided interfaces to the environment.

2) A list of required interfaces from the environment.

3) External specifications.

4) Executable code.

5) Validation code.

6) Design (i.e. documents and source code).

This thesis focuses on all of the aforementioned component internals as they relate to a functionally correct design.

2.4 Module Design

Shared library design is synonymous with module design with respect to this thesis. The purpose of module design is to provide a reusable library of logically related classes who collaborate to expose a given set of functionalities. Since modules are reusable entities, they may have external dependencies which must be satisfied at runtime. The purpose of creating a module is to decompose and group objects within the system for requirements traceability.

2.5 Object Design

Object design is a software abstraction technique whose focus is data encapsulation and real world modeling. This abstraction is realized by localizing data within the system by associating the relevant data with operations defined at the class level. By associating data only with specific operations, the designer can simplify the overall complexity of the respective system. Furthermore, since an object has individual state, it also has a unique identity to identify it despite state changes Kiziltan et al. (2000). This notion lends to the natural flexibility of data abstraction to decompose a software system.

2.6 Operation Design

Operations represent the lowest level of design abstraction in a software system. The operation is responsible for the representation a business requirement(s). At the operation level, design flexibility is limited by imposed restrictions such as: naming schemes, identifier uniqueness, and data sharing (Ward, 1989). Data sharing is the most important restriction in the operation design process. To provide variable traceability, the data values in the operation should only be accessible only via internal operations (Ward, 1989). Global variables have been shown to increase the complexity of the system by introducing unintended data manipulation by external operations Khan et al. (1995).

2.7 Research Methodology

The research for this thesis consisted of reading numerous published studies on software component development and applying them at the language level. Over the course of the research, the following key questions are addressed:

1) Is Component-Oriented development possible at the language level?
2) What levels of component isolation are necessary and plausible?
3) How will component reuse be accomplished?
4) What language level constructs are required to represent a component?
5) At the language level, are components beneficial to the software development process as compared to traditional Object-Oriented techniques?

2.8 Description of Research Methodology

Component-Oriented development poses numerous theoretical and practical key questions for the design and implementation of such a model at the language level. This thesis addresses each question from a pragmatic standpoint to cast light on the ramifications of employing a component model in typical software development.

Component integration at the language level is a critical restriction to mass adoption of CO practices. To realize components at the language level, the component constructs must be strongly typed and conducive to the language itself. Furthermore, the methodology about which components are created and manipulated must support the benefits of the native language and accept the shortcomings as well. By following these basic principles, components can be successfully integrated at the language level.

As with any software model, CO development must exhibit a unique purpose. That purpose must improve the state of software development as we know it today. One such goal is the isolation of the componentry from the remaining system. Component isolation is a set of rules that govern the way the components are accessed and operated on with respect to the global system. Here OO principles can be leveraged to provide the foundation for logical isolation. Within the components, classes will be assigned access and operation modifiers to govern their visibility outside the scope of the enclosing component.

Various component models have existed in the software industry and been proven successful while others have failed. For example, COM is a failed component model outside of internal Microsoft developers due to interoperability issues while J2EE is a successful component model used by thousands of corporations today Sullivan, Marchukov, and Socha (1999). The common goal the component models share is software reuse to usher in a component manufacturing age. Component reuse is a

software engineering concept where verifiable components are reused to provide proven solutions for reoccurring problems. Again, OO principles may be incorporated to dictate the reuse hierarchy and composition strategy. Thereby, component reuse may result from component extension and aggregation with respect to other logically related components.

The intent of this thesis is to analyze whether component constructs at the language level are viable. To represent a component at the language level, a compiler must be written to perform the lexical analysis and grammatical parsing. This task is best accomplished with a pre-compiler that is bootstrapped to the standard java compiler. By using the bootstrapping technique, the compiler integration can be seamless and practical. Bootstrapping would also allow access to existing libraries that are standardized in the Java language.

Software component research is relatively new to the field of computer science and thus lacks extensive scientific study. As such, much anecdotal research has been conducted on the theory of Component Oriented (CO) development yet little scientific verification of those data has been performed. Thus, this thesis supports its assertions by conducting a software metrics analysis. Here, an OO application will be the control in the testing process. The application will then be transformed into a Component-Oriented application representing the variable in the process. All measurable improvements and perceived shortcomings will be documented during the transformation from OO to CO by software measurement.

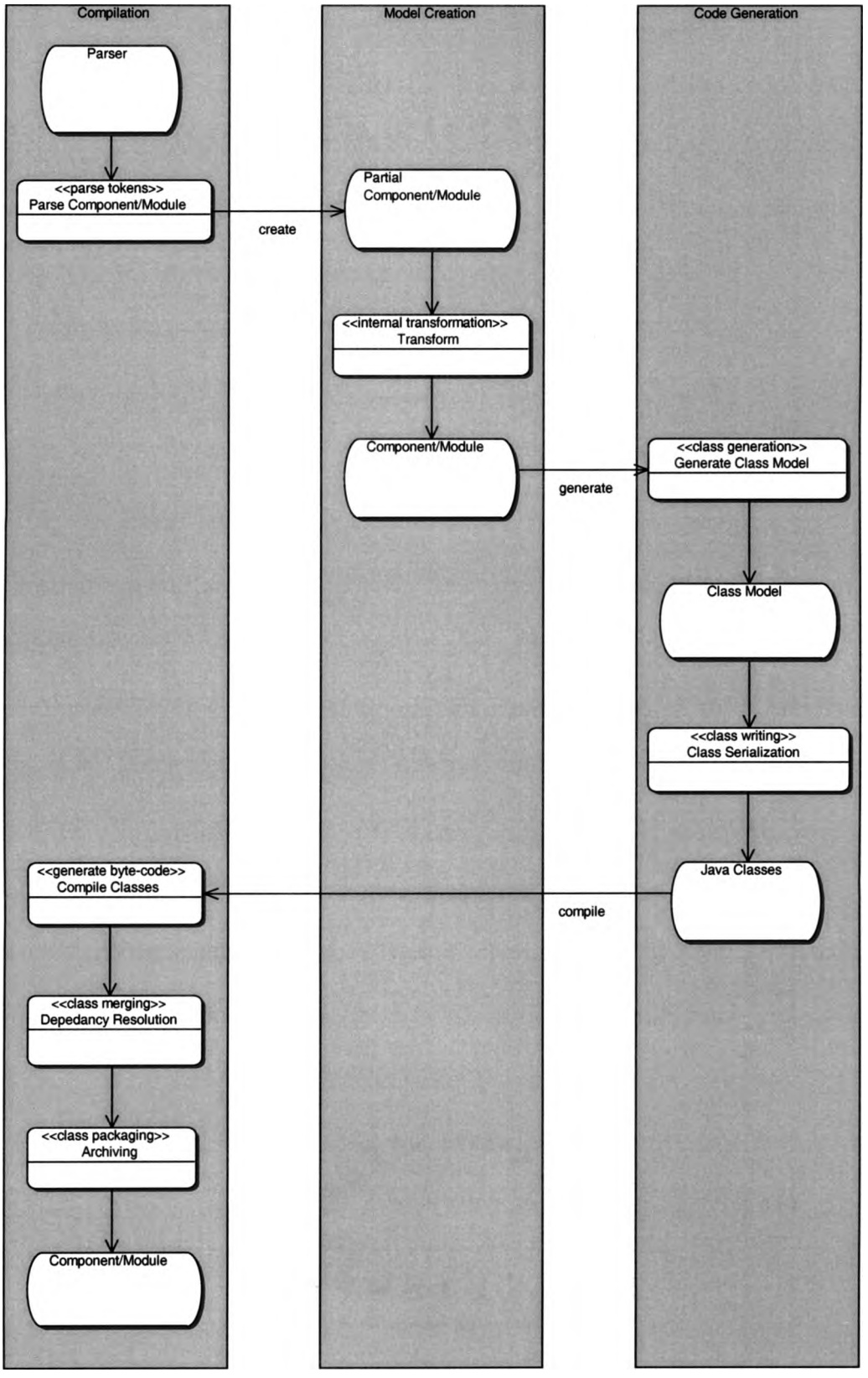
CHAPTER 3

3.1 JLego Introduction

JLego is the use case driven component and module framework, code generator, compiler, runtime, and integration tool presented in this thesis. The name, JLego, was chosen due to the theoretical construction of software with component or Lego like assembly. The goals of JLego are to provide seamless integration with the Java programming language, object and module encapsulation, and contract based design through interface definitions. Within these goals lie various challenges inherent to all component-based systems such as overall complexity and ease of integration with existing Object Oriented systems. JLego attempts to manage the given challenges through continuous Use Case and design refactoring.

The activity coordination with JLego is highly orchestrated from the initial parsing to the end products, components and modules. The activity diagram in figure 3.1.1 depicts the interaction between the various aspects of JLego.

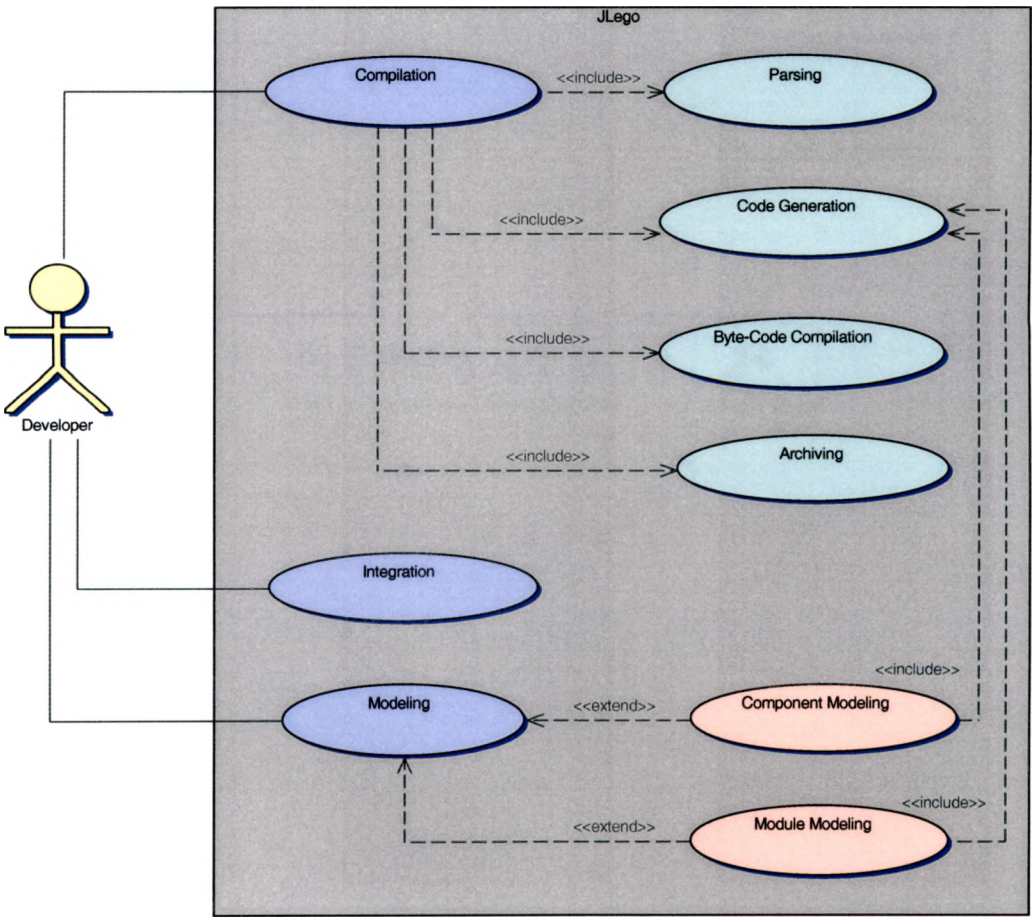
Figure 3.1.1


Compilation
Model Creation
Code Generation
Parser
<<parse tokens>>
Parse Component/Module
create
Partial
Component/Module
<<internal transformation>>
Transform
Component/Module
generate
<<class generation>>
Generate Class Model
Class Model
<<class writing>>
Class Serialization
Java Classes
compile
<<generate byte-code>>
Compile Classes
<<class merging>>
Depedancy Resolution
<<class packaging>>
Archiving
Component/Module

3.2 Use Case

JLego is designed to primarily interact with software developers who are building a Component-Oriented system. Therefore, the command-line tools are designed to offer superb flexibility with minimal disruption. Thus, the major functionalities are integrated into a single tool, a compiler. By integrating the various functionalities, novice component developers can learn the new system quickly and seasoned developers can rapidly compile components. The explicit and derived use case functionalities for the compiler are illustrated in figure 3.2.1.

Figure 3.2.1



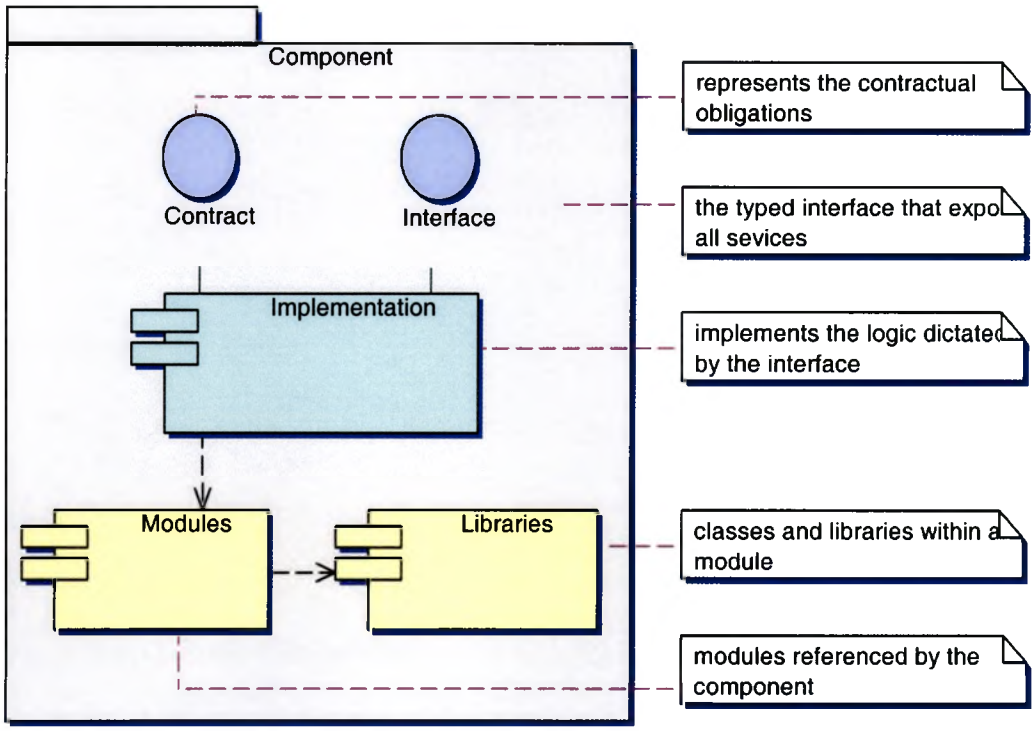
To provide the use case functionalities, JLego incorporates well known design patterns for behavioral realization. Patterns involve a general description of a recurring solution to a recurring problem Kiziltan et al. (2000). In this thesis, patterns provide universal solutions to reoccurring problems in an intuitive fashion.

3.3 Design Model

3.3.1 Component Design

The component design consists of a meta-data model used to represent a software component in Java. Java is the ideal language for a component model for its type-safe grammar to avoid the component vendor blame game whenever a component fails and the inherent platform neutrality Kiziltan et al. (2000). Once a component model is built, the model can be transformed into a tangible component or printed to a text stream in original form. This provides unified compatibility within JLego and associated tools with an emphasis on interface definition and reuse. JLego components use the extension CAR (Component Archive) for filename uniqueness. Figure 3.3.1.1 portrays the inner details of a component.

Figure 3.3.1.1



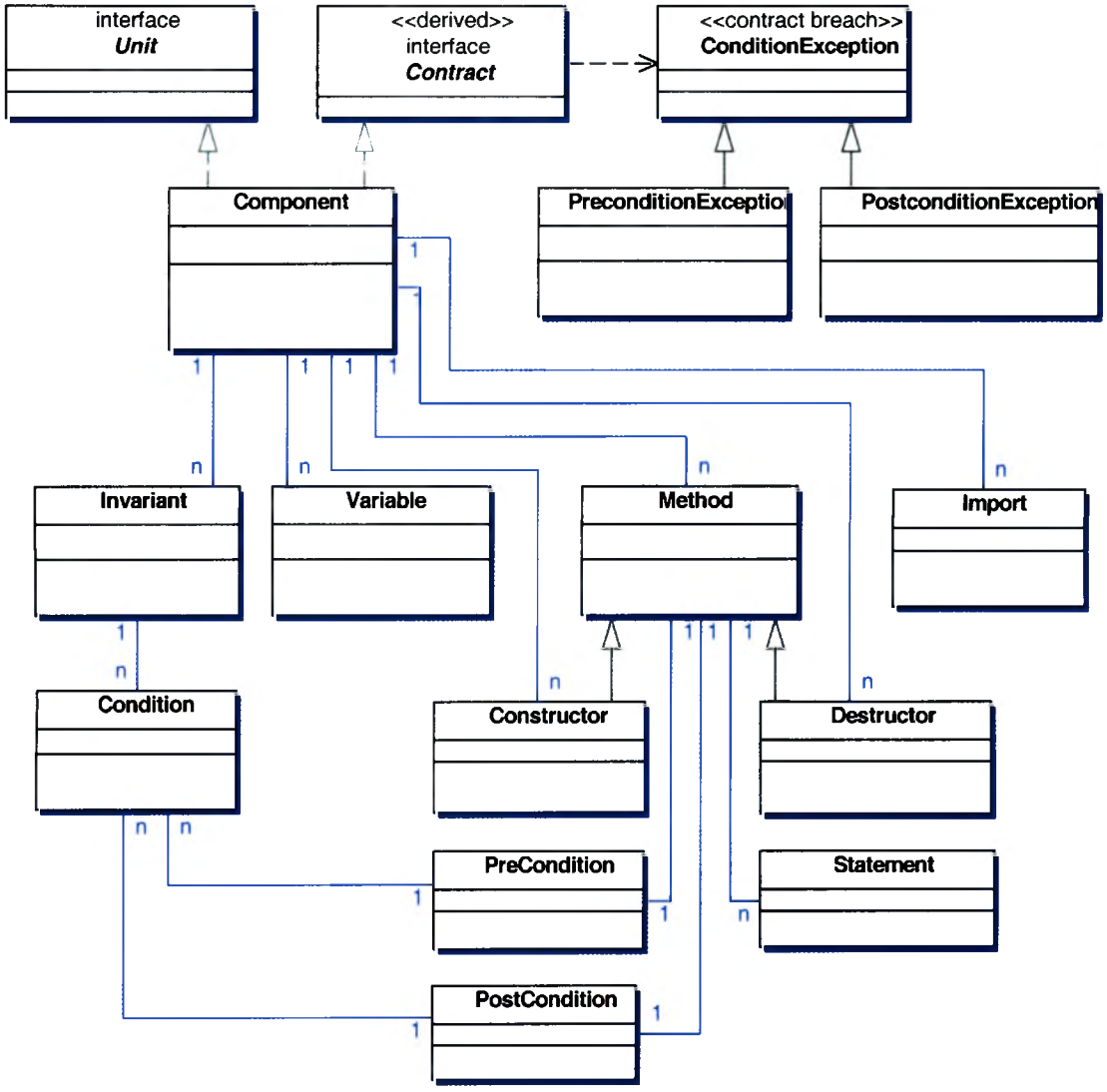
Components in the JLego framework are modeled around the concept of an interface. An interface is a binary structure in the address space of the component whose layout is defined by the rules of the Java language and the JLego model Sullivan et al. (1999). Specifically, the interface exposes the strongly typed, public operations of the component. Only through the interface may a third party component or module conduct contractually bound business Sullivan et al. (1999).

Components can depend on other components Kiziltan et al. (2000). JLego provides such extension methods for adding features to an existing component or module. Since it is not desirable to use the source code of a component as its

specification, a looser coupling via composition instead of inheritance is preferable Kiziltan et al. (2000). The methods are based on the white box inheritance model to abstract the component or module into a white box invariant (Edwards, 1997). To extend a component, a module or component must be aggregated into a proxy component. The proxy component will then mediate the operations performed on the base component by implementing the contract of the base component. The proxy component will appear as a superset of the operations available on the base component and thus provide backwards compatibility.

The component model is represented by a hierarchy of inter-related meta-classes. The root class is named *Component* and has associates: *Constructor*, *Destructor*, *Method*, *Invariant, and Import*. The hierarchy is depicted in figure 3.3.1.2.

Figure 3.3.1.2

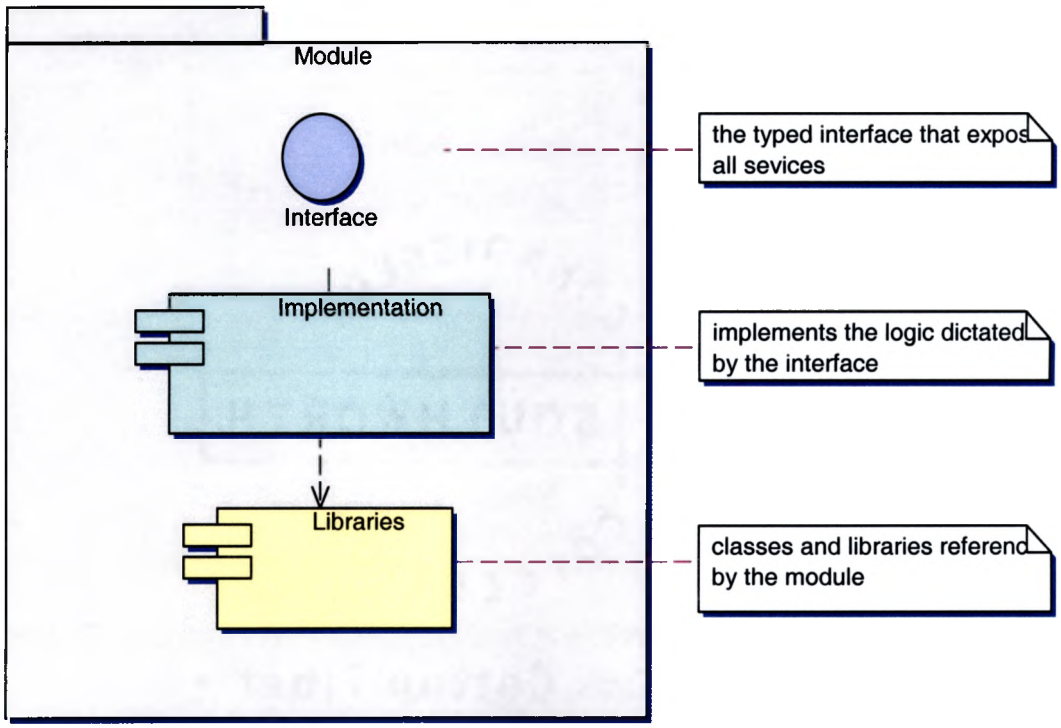


The design-by-contract design pattern is represented as the *Contract* derived interface. If the component interface contract is violated, according to the conditional rules, a *ConditionException* is thrown. The throw presents a contract violation notification to the caller and suspends further execution as a consequence.

3.3.2 Module Design

The module design is a meta-model used to represent a software module in Java. The model is forwards and backwards compatible within the scope of the framework. Thus, a module can be printed to a text stream or transformed into a functional module. JLego modules use the extension MAR (Module Archive) to provide file extension uniqueness. The inner view of a module is shown in figure 3.3.2.1.

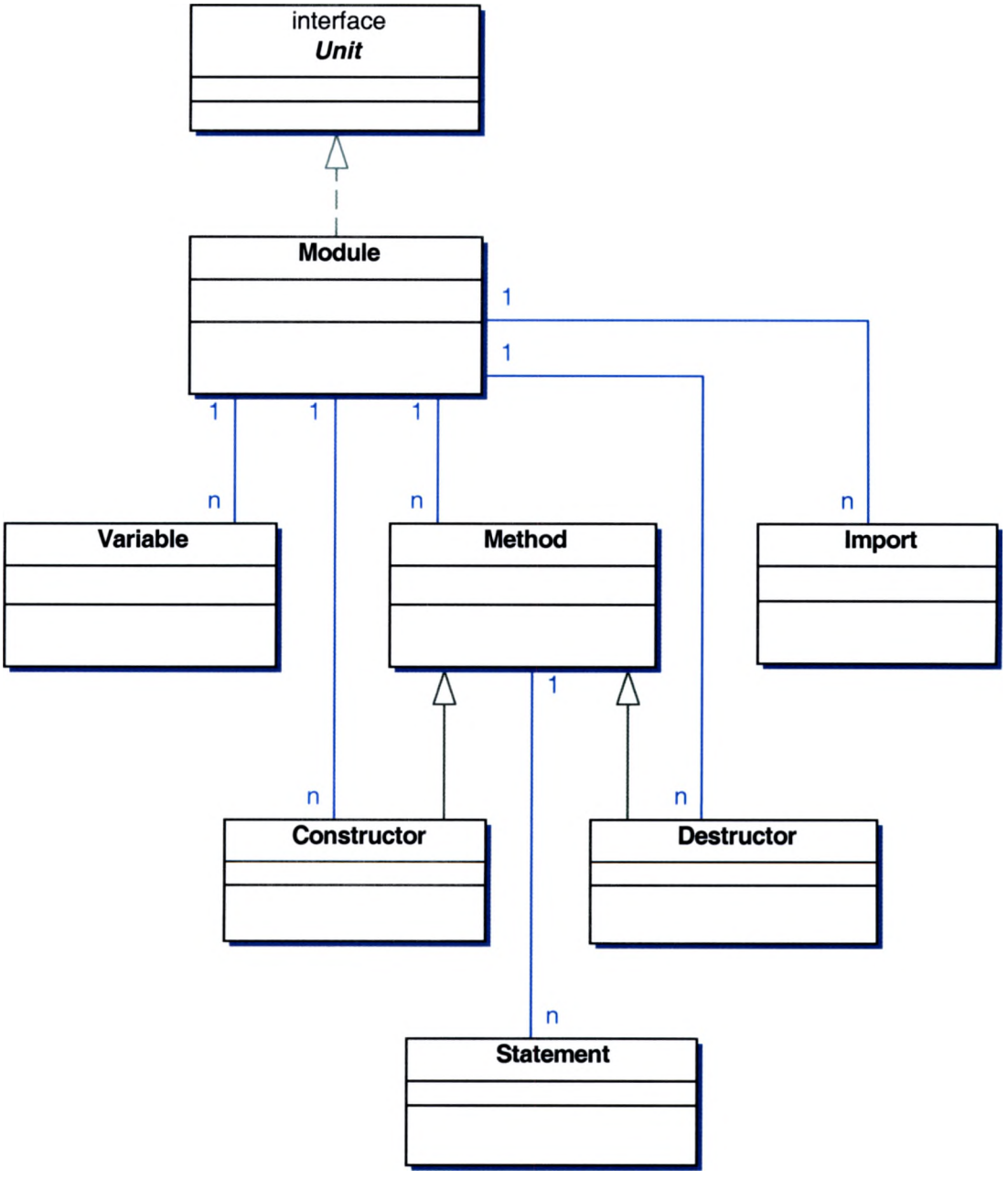
Figure 3.3.2.1



A module is represented by a simple hierarchy of meta-classes. Similar to the component model, the module has a *Module* as the root class, and associates

Constructor, *Destructor*, *Method*, *and Import*. The module class hierarchy is illustrated figure 3.3.2.2.

Figure 3.3.2.2



3.3.3 Component Contract Design

The design-by-contract pattern is a well understood and accepted interaction framework for decoupled systems. JLego components expose the design-by-contract design pattern through a well defined interface that represents the exposed methods of the respective component. The contract is the well defined interface where all input and parameters are validated against an acceptable range. If any of the parameters are outside of the contractual range, the contract is breached and future interaction is terminated. Thus, the contract becomes an integral part of what a component represents from the view of an outside entity. The interaction between the source and destination takes place at the method level of the given component. Within the given method, lie contractual pre and post conditions to validate the terms of the contract. Furthermore, during the interaction with the source, component variables may change state which is inherently bound to the fulfillment of the contract. Therefore, component invariants are provided to enforce the full terms of the contract.

Conceptually, a contract is a set of obligations agreed upon prior to interaction between one or more entities. In JLego, an obligation is mapped to a *Condition* that must be met to continue with the interaction. A *Condition* is a boolean assertion which exposes a contract breach. By definition, assertions are logical expressions about the entities in an interface's information model Lui et al.

(2002). This lends to chaining of contractual *Conditions* to form the foundation of the contract, a set of rules.

A pre-condition expresses requirements that any call of the operation must satisfy if it is to be correct Lui et al. (2002). JLego implements the concept of a pre-condition as *PreCondition* which contains method level *Condition*(s). If a pre-condition assertion fails, a *PreConditionException* is thrown which notifies the caller that a contract breach has occurred and execution is halted. Pre-conditions are straight-forward to implement in the Java language by placing the conditions at the beginning of the method.

A post-condition expresses properties that are ensured in return by the execution of the call Lui et al. (2002). JLego implements the concept of a post-condition as *PostCondition* which contains method level *Condition*(s). If a post-condition assertion fails, a *PostConditionException* is thrown notifying the caller that the result of the execution is invalid. Post-conditions are implemented by wrapping the method in a try/catch block and checking the return value during the final block which is guaranteed to be called before the method returns. The challenge in the implementation is to detect the return type of the method itself, if any, capture the return call(s), and validate the post-conditions defined for that method.

Invariants represent the non-changing states that a component variable may be in and are validated by the component contract specification. Internally, the

invariant is viewed as a *representation invariant* such that indirect interaction with the underlying module or class will have a shared effect on the state of the object (Edwards, 1997). Thus, an invariant is a white box inheritance because information must be known about the module or class (Edwards, 1997). JLego implements the concept of an invariant as *Invariant* which contains component variables as *Condition*(s). Invariants must be true at any point in time while the program is executing which requires techniques like variable substitution. JLego handles the variable substitution by employing method getters and access synchronization. For example, if a component variable of type module is part of an invariant condition stating that the module may not be null, the component model will internally transform any reference to the variable into a synchronized getter method that checks all invariants before returning a reference to the module. Thread synchronization is necessary to avoid race conditions during the invariant condition validation. JLego uses the Java language *synchronization* operator to provide safe concurrent access.

3.3.4 Language Semantics

JLego is modeled around components and modules and thus requires the creation of new grammars for integration at the language level.

The JLego component language is a subset of the Java language with the addition of new types defined to represent a component. Component composition is realized by the inclusion of certain subsets of the Java language to ease integration with the existing object model. This inclusion is necessary for

developers to easily reference and incorporate modules and to invoke the exposed operations.

The module language leverages aggregation to include java language constructs. The aggregation is formed from a tight coupling between the module itself and objects encapsulated within the archive. Essentially, a module is an Object-Oriented library with a well defined interface and lifecycle model incorporated to provide a standards compliant archive.

The lexical and grammatical parsing is performed at the component and module language level. The parser grammar is defined using the JavaCC framework as an extension to the standard Java language grammar. The Java language grammar was an obvious choice for a base grammar because a subset grammar could be easily extracted from the full grammar. Once the base grammar is extracted, the component and module grammars are placed on top of the base grammar. The grammar below depicts the top level component and module parsing for imports and package declaration.

```
Unit Compilation() :
{
 org.swt.edu.jlego.model.Package pkg = null;
 Imports imports = new Imports();
 Unit unit;
}
{
 [ pkg = PackageDeclaration() ]
 ( ImportDeclaration(imports) )*
 unit = TypeDeclaration(pkg, imports)
 <EOF>
```

```
  { return unit; }
}

org.swt.edu.jlego.model.Package PackageDeclaration() :
{
  String name;
}
{
  "package" name = Name() ";"

  { return new org.swt.edu.jlego.model.Package(name); }
}

void ImportDeclaration(Imports imports) :
{
  String name;
  StringBuffer buffer = new StringBuffer();
}
{
  "import" name = Name() { buffer.append(name); } [ "." {
buffer.append("."); }  "*" { buffer.append("*"); } ] ";"

  { imports.add(new Import(buffer.toString())); }
}
```

3.4 Parser

JLego utilizes a module and component parser to validate and build the respective meta-models. The meta-models are created by a lexer and grammatical parser generated by the Java Compiler Compiler (JavaCC). To generate the parser, a grammar is passed to JavaCC and the resulting parser Java source files are generated. Within these files are the actual classes used for tokenizing and parsing the component and module languages.

The generated JavaCC parser can be accessed in two different ways: token traversal, or event model. JLego takes the event model approach for improved overall clarity and the elimination of complicated switch statements for token ID interpretation during token

traversal. The event driven implementation is fairly straightforward in the JavaCC environment. The developer must follow the event flow of tokens through the system and capture the relevant tokens as necessary. Since most of the parsed code is composed of statements, the amount of token capturing work is reduced in the event model. An example of such capturing is demonstrated in the *ComponentDeclaration* section below.

```
Component ComponentDeclaration(org.swt.edu.jlego.model.Package pkg,
Imports imports) :
{
 Component comp = new Component();
 comp.setPackage(pkg);
 comp.setImports(imports);

 is_module = false;

 Token name;
 Composition composition;
}
{
 "component" name = <IDENTIFIER> { comp.setName(name.image); }
 [ "implements" composition = CompositeNameList()
 {comp.setComposition(composition); } ]
 ComponentBody(comp)

 { return comp; }
}
```

The full JavaCC grammar is available for review in Appendix I.

The component and module parser is bootstrapped to the standard Java compiler to leverage the existing Java compiler framework. As a consequence, JLego inherits type checking responsibilities such as class casting and variable scope resolution.

3.5 Meta Transformation

Meta transformation is the process of transforming a meta-component or module into a tangible component or module in the Java programming language. The resultant component or module is directly compilable by the standard Java compiler available in the Java Developer Kit (JDK). There are numerous issues related to the transformation from the meta-model to valid Java source code. JLego attempts to handle these issues in a reasonable manner with respect to the limitations of the Java language.

The meta-model referenced in the transformation is constructed during the parsing phase of the component or module compilation. Since the meta-model has a one-to-one mapping to the meta-component or module, the transformation has a one-to-one mapping with the resultant component or module. The meta-model Java source code is listed in Appendix H.

JLego imposes type checking restrictions on the component or module during the meta-transformation. As with Java classes, JLego components and modules must exist in a file and package with the same name as the component or module. Furthermore, modules and components may not be nested to enforce a one-to-one mapping between requirements and the respective components and modules. These restrictions are imposed to enforce standards based component development. The goal of such standards is to increase component and module reuse in future software projects.

The issues involved in the transformation from meta-model to Java source code revolve around the Java language itself. One such issue is the use of ambiguous import statements within a module that references classes in other packages. During the import transformation phase, multiple classes may match the class listed in the module. JLego handles this possible error by deferring the actual class type checking to the standard Java compiler.

Another issue is the implementation of the meta-post-condition in the resultant component. To be a valid post condition, the return type of the method must be compared to the stated valid range for that return type. Thus, the post condition implementation must properly account for multiple returns and exception handling defined in the given method. For example, a method could throw a user-level exception at any point in the method thus bypassing the post condition range checking. To handle these abnormal scenarios, the standard type-checking mechanism is employed via the *try-finally* clause to correctly validate the return type.

The most obscure issue regarding the transformation process is the implementation of an invariant within the component model. By definition, an invariant must always be valid during the execution of the software. Many practitioners feel that run-time checking of representation invariants is critical to for verification enforcement (Edwards, 1997). Thus, an invariant must be checked at every possible access point and be thread-safe during testing. The primary

strategy for using testing to enforce representation inheritance restrictions is to (Edwards, 1997):

> 1) Use run-time checking to operationally test representation invariants at all necessary points during testing.
>
> 2) Expand white box testing techniques to generate test cases that stress these run-time checks such as synchronization issues.

JLego handles the first requirement by replacing every reference to the given variable with an accessor method generated in the parent component. The accessor method is responsible for checking the given variable before being returned to the caller for active use in the execution path. If the given variable is not valid according to the predetermined invariants, an *InvariantException* is thrown which halts execution in the current thread to realize a contract error. The latter requirement, thread safety, is fulfilled by synchronizing the accessor method for the given variable. Hence, all possible race conditions are satisfied with respect to the accessor method. The limitation to this approach is the invariant could possibly be invalidated in one access and not realized until the next access has occurred. The delay for the invariant contract breach notification is acceptable.

3.6 Compiler

JLego provides byte-code compilation support for components and modules at build time. The actual byte-code compilation is performed by the standard Java compiler when invoked through a standard API (Application Programming Interface) provided in

the JDK (Java Development Kit). This process is transparent to the developer for design simplicity and ease of use.

The Java Compiler is invoked after all modules and components have been generated. The ordering is important because modules must be compiled first so that component dependencies are satisfied. A temporary directory is created for each module to store the byte-code in class files. The modules are then compiled referencing the *Classpath* from the *–classpath* command line option. After the modules are compiled, a *Classpath* for the components is built based upon the *–classpath*, *-buildpath*, and internally compiled modules. Then all of the components are compiled in unique temporary directories. At this point, the modules and components are assumed to be successfully compiled in unique temporary directories.

The compiler's Java source code is listed in Appendix G for reference.

3.7 Archive

A component is an independent unit of deployment where all dependencies are satisfied at runtime. This requirement necessitates the creation of an archive format that resolves dependencies between components and modules. JLego defines such an archival format using standard methodologies. The formats follow the JLego module and a component archive specification.

The component archive specification requires all classes related to the component be packaged in a file using ZIP compression with a "car" extension. Furthermore, all

referenced modules must be included in the component archive to avoid module dependency issues at runtime. To avoid dependency conflicts, a dynamic list of referenced modules and components is created to merge the dependencies. The merge process involves calculating the set difference between the component and its dependencies and copying the set difference of class files into the component. This necessitates the use of static rather than dynamic binding because dynamic binding is handled by the Java Virtual Machine implicitly.

The module archive specification requires all referenced class files and associated libraries be packaged in a file using ZIP compression with a "mar" extension. If any non-referenced classes or libraries are included in the archive, class version mismatches could possibly result. Since modules can reference existing java libraries, dependency issues may result if an incorrect version of the library is present. Thus, JLego employs static binding to resolve those dependencies at compile time rather than runtime. Static binding is implemented by discovering what classes are referenced based upon the import statements in the module and copying the set difference of class files into the module from the libraries. Hence, the libraries are no longer necessary for the module to exist as an independent unit of deployment. Therefore, any components that reference the module can exist as a standalone unit of deployment.

3.8 Runtime

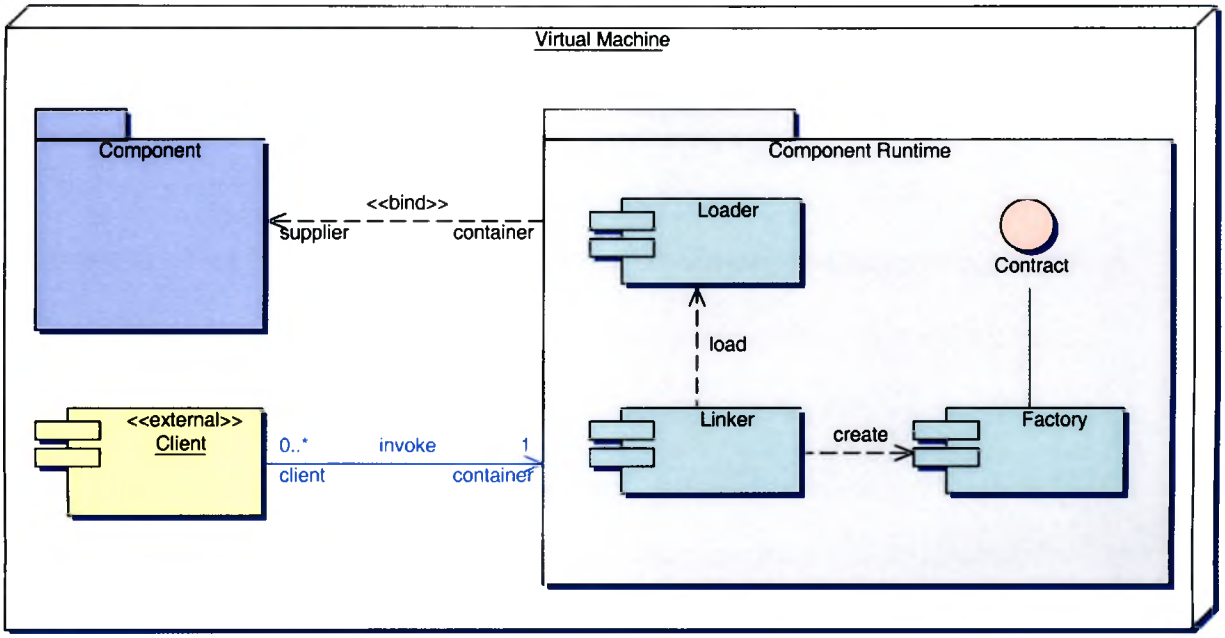
JLego executes under the standard JRE (Java Runtime Environment) distributed by Sun Microsystems. The JRE directly interprets the byte-codes from the class files packaged in the component archive. This is accomplished by including the component in

the classpath of the JRE. The classpath can be set in two different ways: exporting the environment variable CLASSPATH or specifying the –classpath command line option to the JRE.

The JRE uses a linear search to find and load java class files. The search algorithm scans the classpath for possible directories and ZIP archives. If a ZIP archive is found in the classpath, the algorithm includes the class file contents of the archive for potential classloading. JLego adheres to the search process by including all necessary class file dependences at build time. Thus the JRE can directly load the component and all dependant classes in one pass to minimize runtime overhead associated with the components.

The component runtime of JLego compliments the existing Java Virtual Machine (JVM) architecture by using existing classloaders and interfaces to link components. The diagram in figure 3.8.1 illustrates the component runtime model within the JVM.

Figure 3.8.1



At runtime, JLego utilizes the notion of a connector to dynamically bind the component interface to the component implementation. A connector is a protocol specification that defines properties including policies regarding the types of interfaces it can mediate for and certain assurances regarding the non-functional aspect of interaction Kiziltan et al. (2000). When a component's proxy is invoked with an operation, the connector is queried to find the correct implementation for the operation. If the operation implementation is not found, a contract violation is thrown to notify the client.

3.9 Component Integration

To incorporate reusable components into systems, programmers must be able to find and understand them Frakes et al. (1994). JLego emphasizes the need for smooth integration with existing java applications. Integration with new and existing applications is accomplished with standardized naming conventions and classloading

features. JLego employs a standard component loading mechanism that is universally available to new or existing Java programs.

The loading mechanism is based upon the unique instance of the component. For example, a component can support a static loading scheme following the singleton design pattern or require a dynamic loading scheme. By deferring the loading scheme, JLego can easily integrate with an existing system with design adaptation. The loading scheme is further simplified by leveraging the standard Java byte-code class loader to load the components and modules built with JLego. This simplification allows a standard Java application to find a component in the *CLASSPATH*, load the component with the Java class loader, and invoke the exposed services of the respective component. Module loading is transparent to the java application as the modules are packaged within the dependency free component.

The naming conventions associated with a component are based upon the component package, name, and associated methods. The package is used to provide namespace support to avoid possible naming collisions with third party components. The name is used to provide a unique identifier for the component in the same way a class name represents a class. Thus an existing application can get a reference handle to a component named *Car* in package *org.swt.edu.jlego.example.car* with the statement

```
Car car = org.swt.edu.example.car.CarComponent.create(...);
```

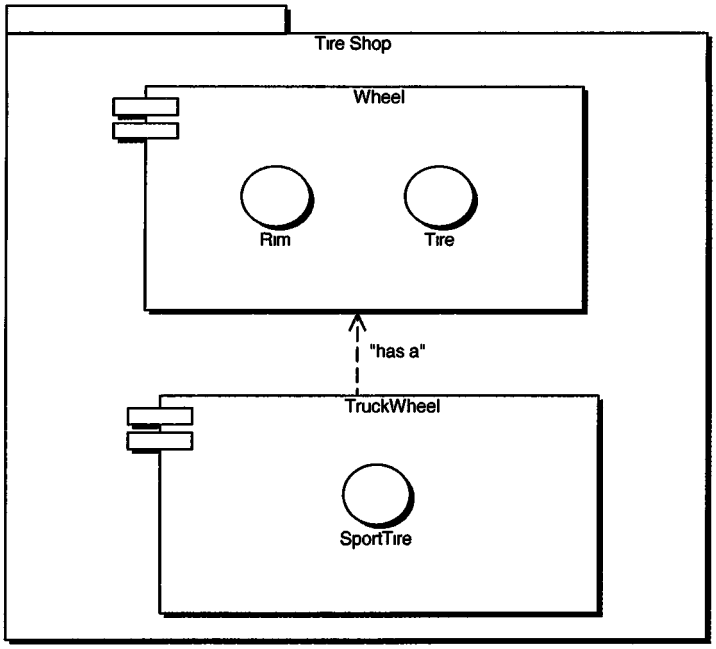
thus treating the component as a normal Java object. Therefore, mandating a standard naming scheme provides consistent and reliable integration between Java objects and components

CHAPTER 4

4.1 Prototype Overview

The JLego system will be demonstrated in the context of a prototype software application. The prototype software application is a tire repair shop that services wheels. The *wheel* is the base component which has a *rim* and *tire*, the base modules. The tire shop also services a *truck wheel*, the extension component, which aggregates its functionality to the base *wheel*. The *truck wheel* adds a *sport tire* module for on-the-fly tire pressure monitoring. The component relationship diagram in figure 4.1.1 illustrates the component and module mappings.

Figure 4.1.1



The purpose of the prototype is to provide a quantitative analysis of a measurable software system using metrics. The verifiable metrics included in the analysis are the result of work by Munson and other notable software metric pioneers such as Halstead (Munson, 2003). From the chosen metrics, a final analysis is performed based upon the hypothesis of this thesis; CO development augments OO development through structured abstraction.

4.2 Component-Oriented Prototype

The prototype is implemented using JLego in a CO fashion to derive a baseline for the analysis. The prototype demonstrates the key benefits of CO development such as behavioral abstraction and deployment independency Lui et al. (2002). The CO prototype is then manually measured using a predefined set of software metrics to quantify the CO abstraction benefits.

The main component in the system is a *Wheel* which is illustrated below.

```
component Wheel {
 module Tire tire;
 module Rim rim;

 invariants{
  invariant(tire.getSize() == rim.getSize(), "wheel size is inconsistent");
  invariant((tire != null) && (rim != null), "rim or tire is null");
 }

 constructor(String rim_brand, int size) {
  pre{
   condition((tire == null) && (rim == null), "modules were previously
initialized");
  }

  post{
```

```
    condition((tire != null) && (rim != null), "modules weren't initialized");
    condition(rim.getBrand().equals(rim_brand), "brand wasn't initialized");
   }

   operation{
    this.tire = load Tire("Michelin", size);
    this.rim = load Rim(rim_brand, size);
   }
  }

  destructor{
   operation{
    this.tire = null;
    this.rim = null;
   }
  }

  method int getPressure(int temperature){
   pre{
    condition((temperature >= 0) && (temperature <= 200), "invalid
temperature");
   }

   post{
    condition((result >= 0) && (result <= 200), "invalid pressure");
   }

   operation{
    return tire.getPressure(temperature);
   }
  }

  /* operations omitted for brevity */

 }
```

One of the modules used in the prototype is a *Rim* as depicted below.

```
 module Rim {
  class CarRim rim;

  constructor(String brand, int size){
   operation{
```

```
    rim = new CarRim(brand, size);
   }
  }

  destructor{
   operation{}
  }

  method int getRotationalAngle(){
   operation{
    return rim.getRotationalAngle();
   }
  }

  /* operations omitted for brevity */

 }
```

The complete component and module source code is available in Appendix E.

4.3 Object-Oriented Result

JLego generates OO source code through the meta-model transformation that conforms to the Java language rules. The object source code is the intermediate result of the component building process and serves as a valid measurement point. The OO metric data provides quantitative information about the transformed components and modules.

The resultant OO *Wheel* component is illustrated below.

```
public interface Wheel {

 public interface Tire extends org.swt.edu.jlego.example.wheel.tire.Tire {}

 public interface Rim extends org.swt.edu.jlego.example.wheel.rim.Rim {}

 public void destroy() throws ConditionException;

 public int getPressure(int temperature) throws ConditionException;
```

```
 public boolean setPressure(int pressure) throws ConditionException;

 public int getSize() throws ConditionException;

 public int getRotationalAngle() throws ConditionException;

}


public final class WheelComponent implements Wheel {

 private org.swt.edu.jlego.example.wheel.tire.Tire tire;
 private org.swt.edu.jlego.example.wheel.rim.Rim rim;

 public static Wheel create(String rim_brand, int size) throws
ConditionException {
  return new WheelComponent(rim_brand, size);
 }

 private WheelComponent(String rim_brand, int size) throws ConditionException
{
  try {
   if(!((tire == null) && (rim == null))){
    throw new PreConditionException("modules were previously initialized");
   }

   this.tire = TireModule.create("Michelin", size);
   this.rim = RimModule.create(rim_brand, size);

   if(!((tire != null) && (rim != null))){
    throw new PostConditionException("modules weren't initialized");
   }
   if(!(rim.getBrand().equals(rim_brand))){
    throw new PostConditionException("brand wasn't initialized");
   }
  }
  catch(ConditionException e) {
   throw e;
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
 }

 protected void finalize(){
```

```
 try{
  destroy();
 }
 catch(Throwable t){}
}

public void destroy() throws ConditionException {
 try {
  this.tire = null;
  this.rim = null;
 }
 catch(Exception e) {
  throw new ConditionException(e.getMessage());
 }
}

public int getPressure(int temperature) throws ConditionException {
 int result = 0;
 boolean has_ex_occured = false;

 try {
  if(!((temperature >= 0) && (temperature <= 200))){
   throw new PreConditionException("invalid temperature");
  }

  result = tire().getPressure(temperature);
  return result;
 }
 catch(ConditionException e) {
  has_ex_occured = true;
  throw e;
 }
 catch(Exception e) {
  has_ex_occured = true;
  throw new ConditionException(e.getMessage());
 }
 finally {
  try {
   if(!has_ex_occured) {
    if(!((result >= 0) && (result <= 200))){
     throw new PostConditionException("invalid pressure");
    }
   }
  }
  catch(ConditionException e) {
   throw e;
```

```
      }
      catch(Exception e) {
       throw new ConditionException(e.getMessage());
      }
     }
    }

    /* operations omitted for brevity */

    private void checkInvariants() {
     if(!(tire.getSize() == rim.getSize())){
      throw new InvariantException("wheel size is inconsistent");
     }

     if(!((tire != null) && (rim != null))){
      throw new InvariantException("rim or tire is null");
     }
    }

    private org.swt.edu.jlego.example.wheel.tire.Tire tire() {
     checkInvariants();

     return tire;
    }

    private org.swt.edu.jlego.example.wheel.rim.Rim rim() {
     checkInvariants();

     return rim;
    }

   }
```

The resultant OO *Rim* module is illustrated below.

```
import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public interface Rim {

 public int getRotationalAngle();

 public int getSize();
```

```
  public String getBrand();

}


import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public final class RimModule implements Rim {

 private org.swt.edu.jlego.example.wheel.rim.CarRim rim;

 public static Rim create(String brand, int size) {
  return new RimModule(brand, size);
 }

 private RimModule(String brand, int size) {
  rim = new CarRim(brand, size);
 }

 protected void finalize(){
  try{
   destroy();
  }
  catch(Throwable t){}
 }

 public void destroy() {
 }

 public int getRotationalAngle() {
  return rim.getRotationalAngle();
 }

 /* operations omitted for brevity */

}
```

The complete OO component and module source code is available in Appendix F.

4.4 Testing Methodology

The software metrics incorporated into the measurement process are

1.) CC - Cyclomatic Complexity
2.) LOC - Lines of Code
3.) NOA - Number of Attributes
4.) NOC - Number of Classes
5.) NOCON - Number of Constructors
6.) NOIS - Number of Import Statements
7.) NOM - Number of Members (Attributes + Operations)
8.) NOO - Number of Operations
9.) n1 - Number of Operators
10.) n2 - Number of Operands

The metrics were chosen from their subjective applicability and information variance. To verify the statistical variance of the above metrics, a Principle Components Analysis (PCA) is performed on the observed metric values. PCA is a multi-variant statistical technique used to determine the relative variance a metric provides to the model (Model). From the PCA, we can determine which metrics are valid and invalid for our model. Valid metrics provide a unique source of statistical variance in a software system (Munson, 2003).

To capture the metrics from the component and object implementations, two distinct methods are employed. The component metrics are recorded by manual counting and the object metrics are recorded using Together, an industry standard modeling tool.

The resultant metric data is the key to exposing the statistical difference between a component and object implementation. For a concise representation, the inclusive modules and components are grouped to form surrogate values which are easily compared. The purpose of the surrogate value is to aggregate the complexity of the metric data into an intuitive data model.

The final step in the analysis is a statistical module and component complexity ranking based upon the computed Fault Index (FI) gathered from the PCA. The FI metric is the weighted sum of a set uncorrelated attribute domain metrics (Munson, 2003). Thus, the FI metric represents the relative complexity of the modules and components.

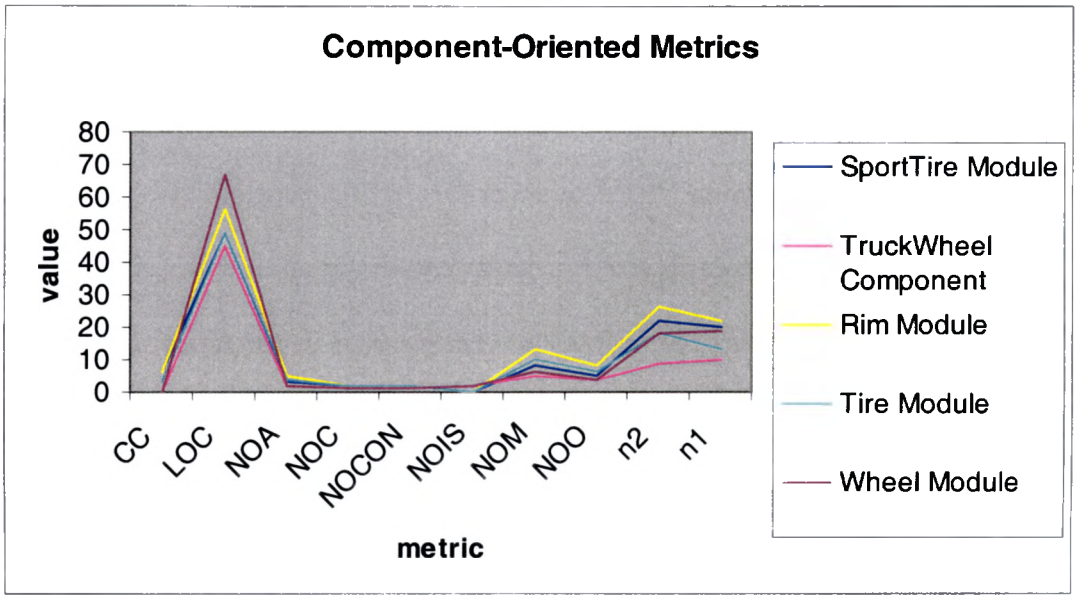
4.5 Testing Overview

The experiment is designed to measure the quantitative benefits of CO development over OO development. The CO prototype is first measured by hand using the chosen software metrics. JLego then compiles the prototype components and modules into OO source code. The OO code is then analyzed by Together producing a tab-delimited metrics table. The CO and OO metrics are then combined via a set union to form a unified table. A PCA is performed on the unified table to extract the valid metrics and compute the Fault Index (FI) for the components and modules. The resultant FI weights are then computed to create weighted charts. Lastly, the CO abstraction benefits are examined by comparing the resultant CO metrics to the sister OO metrics.

4.6 Test Results

The CO metrics table in Appendix A is modeled as a series line chart in figure 4.6.1. The chart shows the metric values for each component and module in the prototype implementation. A higher metric score represents a greater complexity for each component and module.

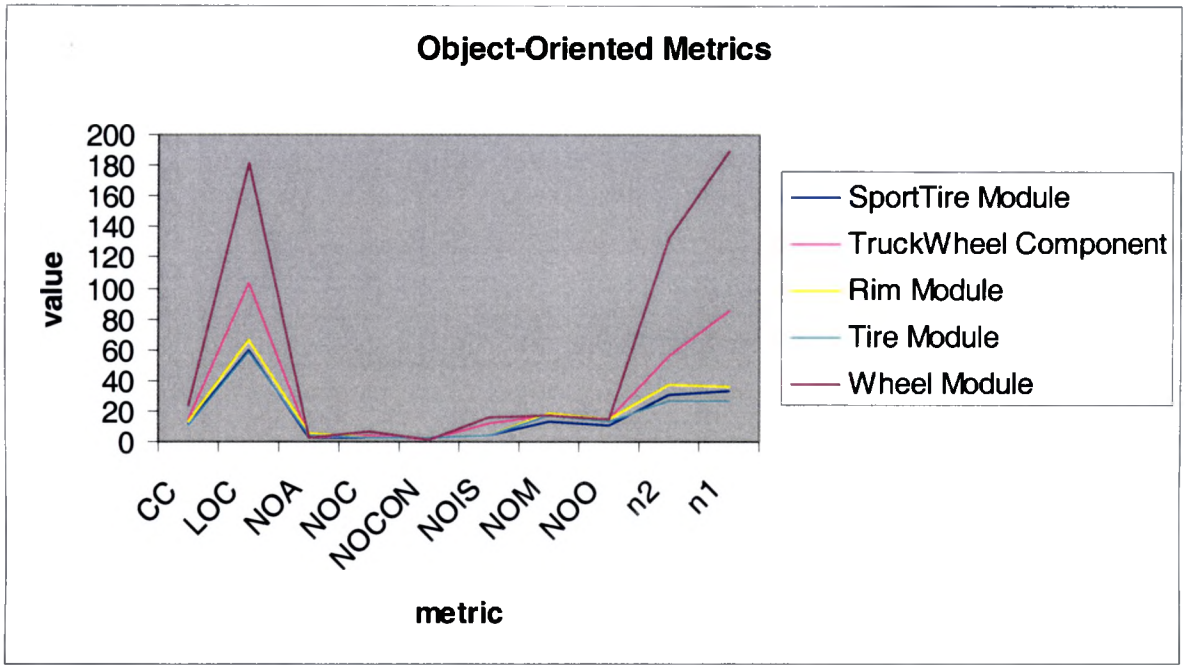
Figure 4.6.1



From the CO metrics chart, one can compare the relative complexity of the components and modules based upon the metric scores. This complexity information begins the formation a reliable CO complexity model using FI metrics (Munson, 2003). For the CC, LOC, and NOA metrics, the *Wheel* component is clearly the most complex compilation unit. Regarding Halstead's n1 and n2 metrics, the *Rim* module is the most complex compilation unit.

The OO metrics chart in figure 4.6.2 represents the OO metrics table in Appendix B. The chart depicts the metric values for the resultant OO component and module implementations. A higher metric score indicates a greater overall complexity for the surrogate component or module class.

Figure 4.6.2

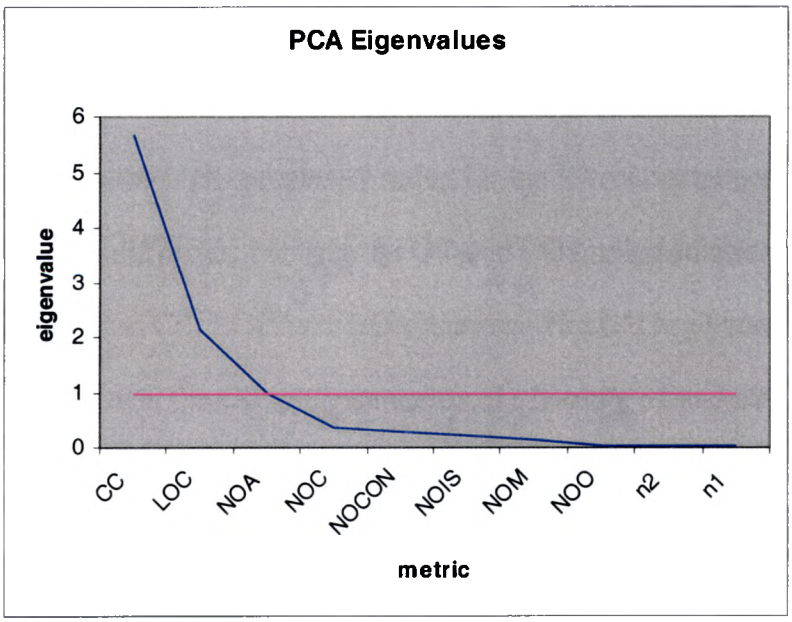


The OO metrics chart conveys the relative complexity of the resultant components and modules. The complexity information begins the formation of a quantitative OO complexity model using FI metrics (Munson, 2003). The CC, LOC, and NOA metrics indicate that the resultant *Wheel* component is the most complex compilation unit. This finding corresponds to the high complexity rating for the *Wheel* component in figure 4.6.1. The observable difference between the OO and CO implementations is the non-linear relationship for the CC, LOC, and NOA metrics. The OO implementation is clearly more complex than the CO implementation. Halstead's n1 and n2 metrics indicate that the resultant *Wheel* component is the most complex compilation unit. This finding doesn't correspond to the *Wheel* component complexity rating in figure 4.6.1 as expected by the researcher. Whenever the CO implementation is transformed into an OO implementation, component services are provided which adds to the total operand and operator count. Notice that the resultant *TruckWheel* component has a higher n1 and n2

complexity than in figure 4.6.1 as a result of the incorporated component services. Thus, the n1 and n2 metrics for the OO and CO components are expected to have a non-linear relationship. In contrast, CO and OO modules abstract few services and as such form a linear relationship which is shown in figure 4.6.1.

The data table in Appendix C is the result of the PCA on the combined OO and CO metrics. The metric eigenvalues in the table are graphed in figure 4.6.3. An eigenvalue represents a proportion of variance explained by the associated metric. The chosen baseline eigenvalue is shown as a purple line in the chart. The baseline is computed from a random data sample where the value is always one (Munson, 2003). A baseline of one is a common stopping rule for determining whether a metric is valid (Munson, 2003).

Figure 4.6.3

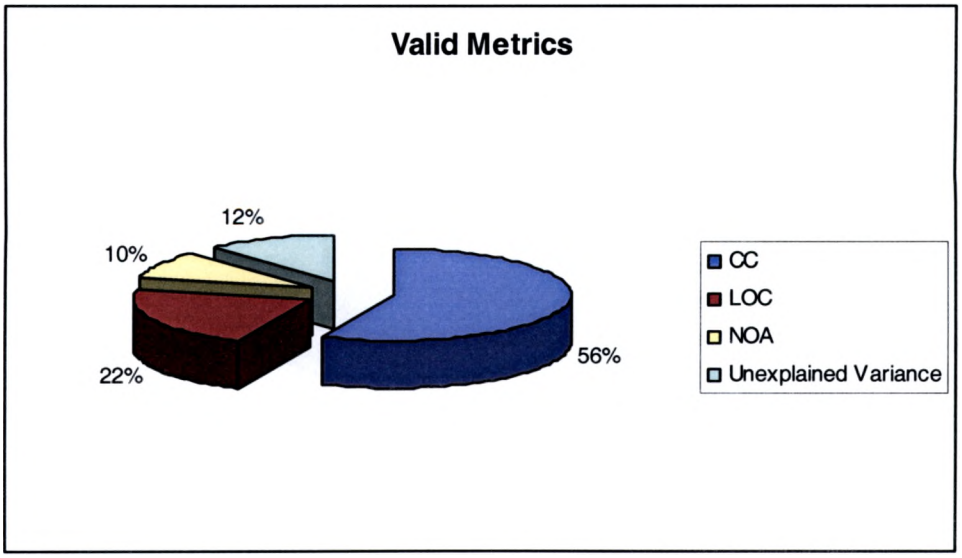


From the chart, we can determine that the CC, LOC, and NOA metrics are valid. The remaining metrics fail to provide a unique source of variance as they fall below our predetermined baseline. By narrowing the scope to statistically valid metrics, we can reduce potential noise in our model (Munson, 2003). From this point, the analysis focus is on the validated software metrics.

4.7 Analysis of Test Results

The valid metrics are derived from the PCA analysis shown in figure 4.6.3. In a given system, the metrics should account for at least 80% of the total variance to be statistically accurate (Munson, 2003). To verify our valid metrics meet the minimum criteria, a pie chart is constructed in figure 4.7.1 showing the relative percentage of variance coverage.

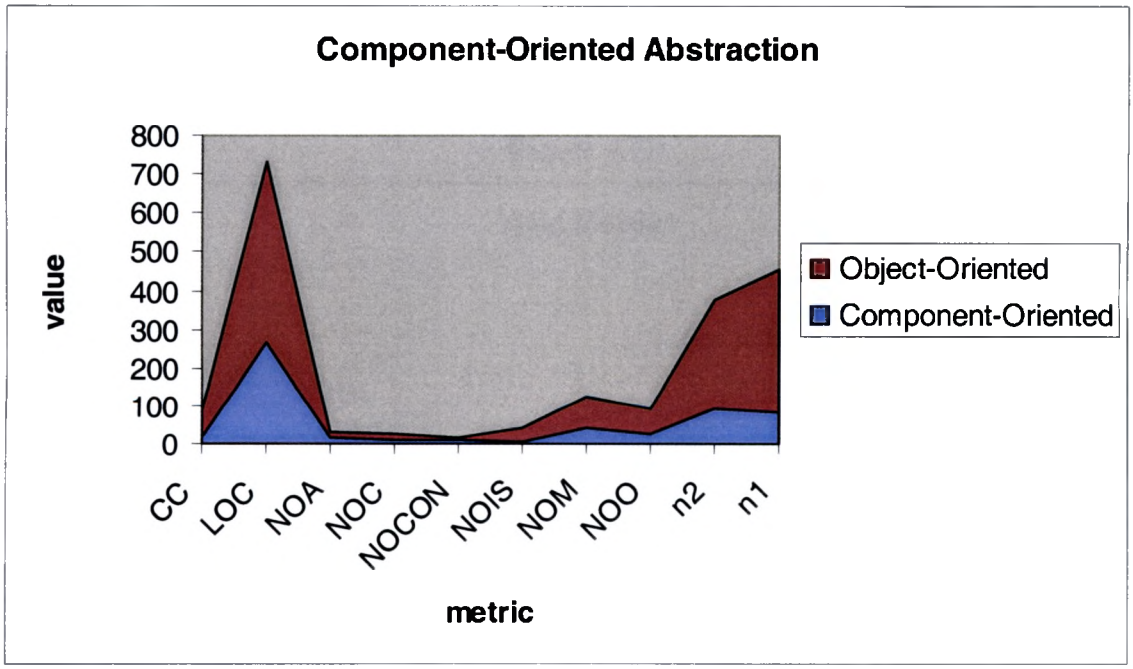
Figure 4.7.1



The valid metrics chart shows that the combined valid metric coverage is 88% of the total system variance. This confirms that the resultant metrics are statistically accurate for the remaining analysis.

The validation of this thesis' hypothesis lies in the proof of component abstraction. In figure 4.7.2, an area chart is illustrated to show the complexity for the OO and CO implementations. The metric values for each data series are the cumulative project metrics for the respective CO and OO implementations. The area under each series then signifies the overall complexity of the respective implementation.

Figure 4.7.2

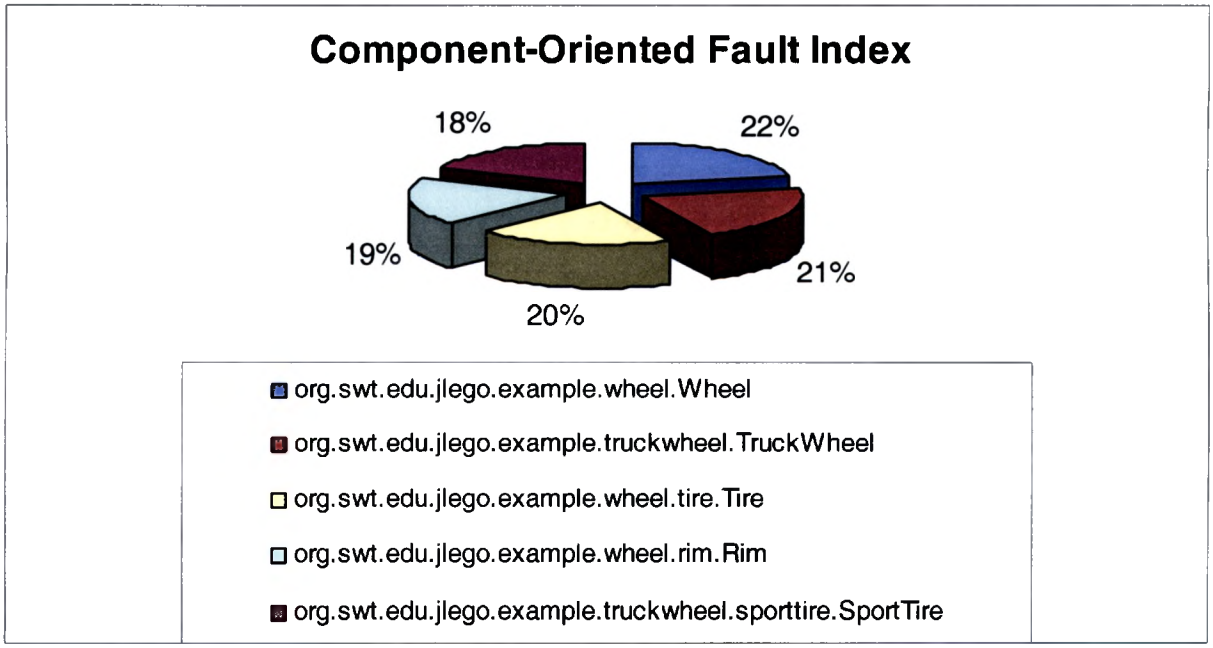


From the component abstraction chart, one can surmise that the prototype CO implementation is significantly less complex than the resultant OO implementation due to

the 40% area difference. The explanation for the non-linear complexity displacement is directly related to the services abstracted by JLego. If the component services were removed, the complexity would form a linear relationship. Therefore, one may conclude that language level CO development provides verifiable software abstraction benefits.

An important aspect of software abstraction is to maintain quality assurance at each abstraction level. Static compilation unit complexity is represented by the surrogate FI metric in this experiment. The table in Appendix C shows the FI for each component and module in the prototype. From the FI table, the CO FI index is illustrated in figure 4.7.3 as a risk percentage relative to the total system risk. High FI values indicate a greater likelihood of discovering faults in the respective module or component (Munson, 2003).

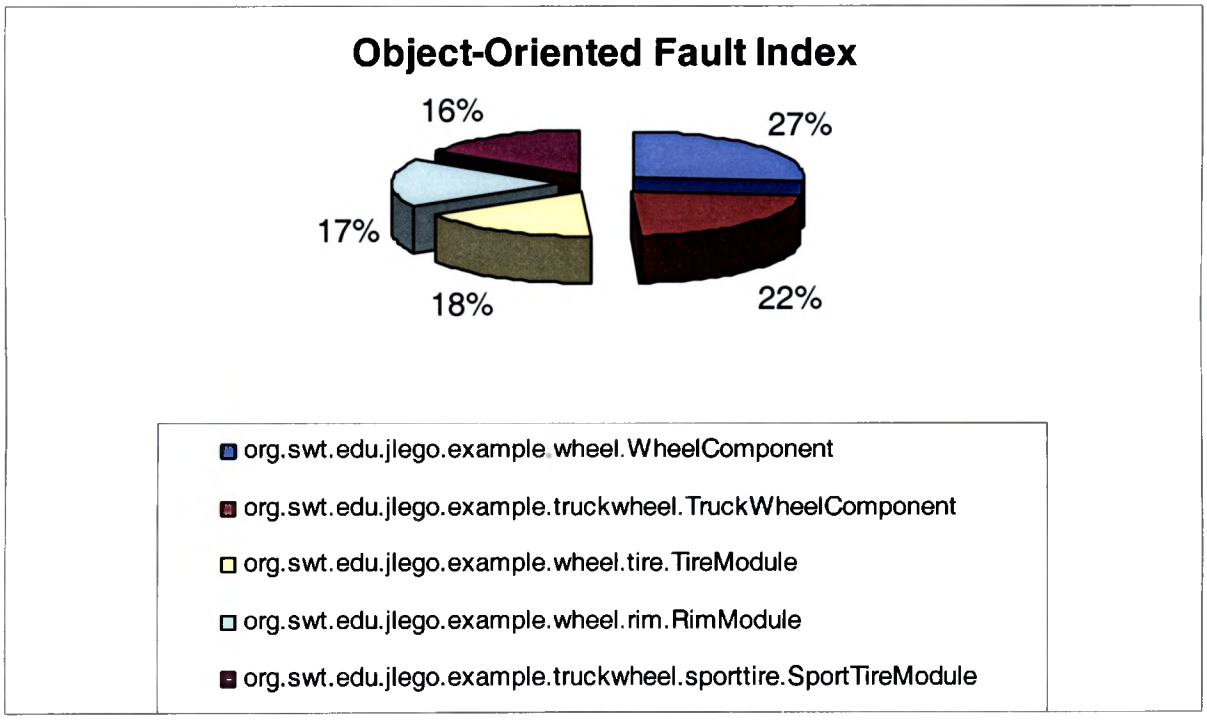
Figure 4.7.3



The CO FI chart lists the relative risk percentage of each module and component where the *Wheel* component has the greatest FI value. As expected, the FI for the components is higher than the FI for the modules by a small percentage. The standard deviation for the risk distribution is ±4% indicating a homogenous complexity distribution.

For the OO FI analysis, the table in Appendix C is referenced to compose the chart in figure 4.7.4. The composite OO FI chart illuminates the risk percentage of the resultant Java classes relative to the total system risk. Note that only the Java classes corresponding to the prototype components are depicted.

Figure 4.7.4



The OO FI chart lists the relative complexity of each component and module where the *Wheel* component has the greatest complexity. From the CO abstraction chart in figure

4.7.2, we could hypothesize that the OO FI has a greater standard deviation than the CO FI. This is shown to be the case in the OO FI chart where the standard deviation is ±11% over the entire risk distribution. The measured increase in standard deviation is attributed to the abstracted component services in the OO implementation. Another observable fact is the complexity ordering between the CO and OO FI. Though the standard deviation has increased in the OO FI model, the complexity ranking of the respective implementations remains linear. Thus, measuring the CO FI can provide a predictive OO FI and vice versa for a given system. This linear relationship could provide statistical value in reverse engineering an existing OO system to qualify the level of abstraction available in an equivalent CO system

CHAPTER 5

5.1 Research Limitations

Numerous limitations were encountered while researching the parser framework and implementation. One such limitation is the lack of full validation during the translation phase of the compilation process in JLego. Another limitation is the inability to properly load a component within a third party Java class using optimal semantics.

The compilation strategy employed is similar to a pre-compiler that expands macros into valid source code. The translation assumes the responsibility of language verification and validation. The verification is accomplished through lexical and type checking performed by the pre-compiler, but source code validation is deferred to the actual language compiler. The result is an intermediary step where macro validation does not occur, hence obfuscating the resultant errors after the meta-expansion and compilation are performed. The solution to the validation issue is to implement full validation checking in the pre-compiler. Then, validation errors before the macro expansion would be correctly detected and displayed.

The semantics of loading a component are not ideal in their present state. Currently, the standard class loading mechanism is used through the *new* keyword, but ideally a keyword such as *load* should be available in the standard java compiler to load

components. The solution to this issue would be to implement the load keyword in the Java compiler or create a bootstrapped java compiler that would perform pre-compilation on the Java source code replacing the keyword *load* with a component reference.

5.2 Future Research

The language level component model presented in this paper is designed to be both extensible and scalable. Some possible extensions to the system include: remote object transport and method invocation, java build environment integration with Ant, source code syntax highlighting in an Integrated Development Environment (IDE) such as JBuilder, and integration with existing languages like C++ or Python.

5.3 Conclusion

As of this writing, no existing programming language supports Component-Oriented development at the language level. This research asserts that language level components are practical and capable of being seamlessly integrated with existing software. By leveraging componentry at the language level, the design goals of modern software can be achieved. The Component-Oriented (CO) approach provides a method of functional abstraction found in structured design yet missing in object design. To this end, components decompose the design architecture into a manageable set of independent, structured abstractions thus allowing humans to better comprehend the system. Therefore, CO development is necessary for the natural progression of software development into an engineering discipline.

APPENDICES

Appendix A

Component-Oriented Metrics										
Compilation Unit	CC	LOC	NOA	NOC	NOCON	NOIS	NOM	NOO	n2	n1
org.swt.edu.jlego.example.truckwheel.sporttire.SportTire	0	24	1	1	1	0	3	2	4	2
org.swt.edu.jlego.example.truckwheel.sporttire.PressureSensor	6	25	2	1	1	0	5	3	18	18
SportTire Module	6	49	3	2	2	0	8	5	22	20
org.swt.edu.jlego.example.truckwheel.TruckWheel	0	45	2	1	1	2	5	4	9	10
TruckWheel Component	0	45	2	1	1	2	5	4	9	10
org.swt.edu.jlego.example.wheel.rim.Rim	0	27	1	1	1	0	4	3	3	2
org.swt.edu.jlego.example.wheel.rim.CarRim	6	29	4	1	1	0	9	5	23	20
Rim Module	6	56	5	2	2	0	13	8	26	22
org.swt.edu.jlego.example.wheel.tire.Tire	0	32	1	1	1	0	5	4	7	5
org.swt.edu.jlego.example.wheel.tire.CarTire	3	17	3	1	1	0	5	2	11	8
Tire Module	3	49	4	2	2	0	10	6	18	13
org.swt.edu.jlego.example.wheel.Wheel	0	67	2	1	1	2	6	4	18	19
Wheel Module	0	67	2	1	1	2	6	4	18	19
Project	15	266	16	8	8	4	42	27	93	84

Appendix B

Object-Oriented Metrics										
Compilation Unit	CC	LOC	NOA	NOC	NOCON	NOIS	NOM	NOO	n2	n1
org.swt.edu.jlego.example.truckwheel.sporttire.SportTire	0	7	0	1	0	2	2	2	0	0
org.swt.edu.jlego.example.truckwheel.sporttire.SportTireModule	6	27	1	1	1	2	6	5	13	15
org.swt.edu.jlego.example.truckwheel.sporttire.PressureSensor	6	25	2	1	1	0	5	3	18	18
SportTire Module	12	59	3	3	2	4	13	10	31	33
org.swt.edu.jlego.example.truckwheel.TruckWheel	0	13	0	2	0	4	5	5	0	0
org.swt.edu.jlego.example.truckwheel.TruckWheel.SportTire	0	1	0	1	0	4	0	0	0	0
org.swt.edu.jlego.example.truckwheel.TruckWheelComponent	14	89	2	1	1	4	12	10	55	85
TruckWheel Component	14	103	2	4	1	12	17	15	55	85
org.swt.edu.jlego.example.wheel.rim.Rim	0	8	0	1	0	2	3	3	0	0
org.swt.edu.jlego.example.wheel.rim.RimModule	7	29	1	1	1	2	7	6	14	16
org.swt.edu.jlego.example.wheel.rim.CarRim	6	29	4	1	1	0	9	5	23	20
Rim Module	13	66	5	3	2	4	19	14	37	36
org.swt.edu.jlego.example.wheel.tire.Tire	0	9	0	1	0	2	4	4	0	0
org.swt.edu.jlego.example.wheel.tire.TireModule	8	32	1	1	1	2	8	7	16	19
org.swt.edu.jlego.example.wheel.tire.CarTire	3	17	3	1	1	0	5	2	11	8
Tire Module	11	58	4	3	2	4	17	13	27	27
org.swt.edu.jlego.example.wheel.Wheel	0	14	0	3	0	4	5	5	0	0
org.swt.edu.jlego.example.wheel.Wheel.Rim	0	1	0	1	0	4	0	0	0	0
org.swt.edu.jlego.example.wheel.Wheel.Tire	0	1	0	1	0	4	0	0	0	0
org.swt.edu.jlego.example.wheel.WheelComponent	24	166	2	1	1	4	12	10	133	189
Wheel Module	24	182	2	6	1	16	17	15	133	189
Project	74	468	16	19	8	40	83	67	283	370

Appendix C

PCA Eigenvalues											
	CC	LOC	NOA	NOC	NOCON	NOIS	NOM	NOO	n2	n1	total
Eigenvalue	5.67	2.16	0.98	0.38	0.28	0.22	0.16	0.05	0.05	0.05	10
Percentage	56.70%	21.60%	9.80%	3.80%	2.80%	2.20%	1.60%	0.50%	0.50%	0.50%	100.00%

Appendix D

Principle Components Analysis				
Compilation Unit	Domain #1	Domain #2	Domain #3	FI
Component				
org.swt.edu.jlego.example.wheel.Wheel	0.29861	-0.54388	0.07364	50.95703
org.swt.edu.jlego.example.truckwheel.TruckWheel	0.07383	-0.63376	0.15114	48.69167
org.swt.edu.jlego.example.wheel.tire.Tire	-0.11297	-0.95188	0.26746	46.03520
org.swt.edu.jlego.example.wheel.rim.Rim	-0.26889	-1.01048	0.10331	44.12833
org.swt.edu.jlego.example.truckwheel.sporttire.SportTire	-0.37801	-1.03992	-0.11845	42.66352
Object				
org.swt.edu.jlego.example.wheel.WheelComponent	3.23060	1.52755	-1.33676	83.03094
org.swt.edu.jlego.example.truckwheel.TruckWheelComponent	1.82577	0.68957	0.23102	69.63458
org.swt.edu.jlego.example.wheel.Wheel	-0.63779	1.99586	2.93876	55.84739
org.swt.edu.jlego.example.wheel.tire.TireModule	0.51159	-0.16822	0.43281	54.81833
org.swt.edu.jlego.example.wheel.rim.RimModule	0.34858	-0.22983	0.27114	52.83949
org.swt.edu.jlego.example.wheel.rim.CarRim	0.70295	-1.51707	0.91613	52.60806
org.swt.edu.jlego.example.truckwheel.TruckWheel	-0.57347	1.45682	1.33388	51.97462
org.swt.edu.jlego.example.truckwheel.sporttire.SportTireModule	0.20269	-0.28035	0.08871	51.02427
org.swt.edu.jlego.example.truckwheel.sporttire.PressureSensor	0.16412	-1.09190	0.02956	47.71698
org.swt.edu.jlego.example.wheel.tire.CarTire	0.01541	-1.48116	0.25007	45.32810
org.swt.edu.jlego.example.wheel.tire.Tire	-0.63062	0.36137	-0.30074	44.97531
org.swt.edu.jlego.example.wheel.rim.Rim	-0.73649	0.33205	-0.52135	43.54264
org.swt.edu.jlego.example.truckwheel.sporttire.SportTire	-0.84237	0.30273	-0.74197	42.10996
org.swt.edu.jlego.example.truckwheel.TruckWheel.SportTire	-1.06451	0.76083	-1.35612	40.69120
org.swt.edu.jlego.example.wheel.Wheel.Rim	-1.06451	0.76083	-1.35612	40.69120
org.swt.edu.jlego.example.wheel.Wheel.Tire	-1.06451	0.76083	-1.35612	40.69120

Appendix E

```
package org.swt.edu.jlego.example.wheel;

import org.swt.edu.jlego.example.wheel.tire.*;
import org.swt.edu.jlego.example.wheel.rim.*;

component Wheel {
  module Tire tire;
  module Rim rim;
```

```
 invariants{
  invariant(tire.getSize() == rim.getSize(), "wheel size is inconsistent");
  invariant((tire != null) && (rim != null), "rim or tire is null");
 }

 constructor(String rim_brand, int size) {
  pre{
   condition((tire == null) && (rim == null), "modules were previously
initialized");
  }

  post{
   condition((tire != null) && (rim != null), "modules weren't initialized");
   condition(rim.getBrand().equals(rim_brand), "brand wasn't initialized");
  }

  operation{
   this.tire = load Tire("Michelin", size);
   this.rim = load Rim(rim_brand, size);
  }
 }

 destructor{
  operation{
   this.tire = null;
   this.rim = null;
  }
 }

 method int getPressure(int temperature){
  pre{
   condition((temperature >= 0) && (temperature <= 200), "invalid
temperature");
  }

  post{
   condition((result >= 0) && (result <= 200), "invalid pressure");
  }

  operation{
   return tire.getPressure(temperature);
  }
 }

 method boolean setPressure(int pressure){
  pre{
```

```
    condition((pressure >= 0) && (pressure <= 100), "invalid pressure");
   }

   operation{
    return tire.setPressure(pressure);
   }
  }

  method int getSize(){
   post{
    condition((result >= 0) && (result <= 30), "invalid size");
   }

   operation{
    return rim.getSize();
   }
  }

  method int getRotationalAngle(){
   post{
    condition(result >= 0 && result <= 360, "invalid angle");
   }

   operation{
    return rim.getRotationalAngle();
   }
  }

}



package org.swt.edu.jlego.example.truckwheel;

import org.swt.edu.jlego.example.truckwheel.sporttire.*;
import org.swt.edu.jlego.example.wheel.*;

component TruckWheel {
 component Wheel wheel;
 module SportTire tire;

 invariants{
  invariant(wheel != null, "wheel is null");
  invariant(tire != null, "tire is null");
 }
```

```
 constructor(String rim_brand, int size) throws Exception{
  operation{
   this.wheel = load Wheel(rim_brand, size);
   this.tire = load SportTire();
  }
 }

 destructor{
  operation{
   this.wheel = null;
  }
 }

 method int getPressure(int temperature){
  operation{
   return tire.getPressure(temperature);
  }
 }

 method boolean setPressure(int pressure){
  pre{
   condition((pressure >= 0) && (pressure <= 30), "invalid pressure");
  }

  operation{
   return tire.setPressure(pressure);
  }
 }

 method int getSize(){
  operation{
   return wheel.getSize();
  }
 }

 method int getRotationalAngle(){
  operation{
   return wheel.getRotationalAngle();
  }
 }

}


package org.swt.edu.jlego.example.wheel.rim;
```

```
module Rim {
 class CarRim rim;

 constructor(String brand, int size){
  operation{
   rim = new CarRim(brand, size);
  }
 }

 destructor{
  operation{}
 }

 method int getRotationalAngle(){
  operation{
   return rim.getRotationalAngle();
  }
 }

 method int getSize(){
  operation{
   return rim.getSize();
  }
 }

 method String getBrand(){
  operation{
   return rim.getBrand();
  }
 }
}


package org.swt.edu.jlego.example.truckwheel.sporttire;

module SportTire {
 class PressureSensor sensor;

 constructor(){
  operation{
   sensor = new PressureSensor(28);
  }
 }

 destructor{
  operation{
```

```
    sensor.stop();
   }
  }

  method int getPressure(int temperature){
   operation{
    return sensor.getPressure(temperature);
   }
  }

  method boolean setPressure(int pressure){
   operation{
    return sensor.setPressure(pressure);
   }
  }

}


package org.swt.edu.jlego.example.wheel.tire;

module Tire {
  class CarTire tire;

  constructor(String brand, int size){
   operation{
    tire = new CarTire(brand, size);
   }
  }

  destructor{
   operation{}
  }

  method int getPressure(int temperature){
   operation{
    throw new UnsupportedOperationException("generic tire doesn't support
pressure reading");
   }
  }

  method boolean setPressure(int pressure) throws
UnsupportedOperationException {
   operation{
    throw new UnsupportedOperationException("generic tire doesn't support self-
inflation");
```

```
    }
  }

  method int getSize(){
    operation{
      return tire.getSize();
    }
  }

  method String getBrand(){
    operation{
      return tire.getBrand();
    }
  }
}
```

Appendix F

```
package org.swt.edu.jlego.example.wheel;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.example.wheel.tire.*;
import org.swt.edu.jlego.example.wheel.rim.*;
import org.swt.edu.jlego.*;

public interface Wheel {

  public interface Tire extends org.swt.edu.jlego.example.wheel.tire.Tire {}

  public interface Rim extends org.swt.edu.jlego.example.wheel.rim.Rim {}

  public void destroy() throws ConditionException;

  public int getPressure(int temperature) throws ConditionException;

  public boolean setPressure(int pressure) throws ConditionException;

  public int getSize() throws ConditionException;

  public int getRotationalAngle() throws ConditionException;

}
```

```
package org.swt.edu.jlego.example.wheel;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.example.wheel.tire.*;
import org.swt.edu.jlego.example.wheel.rim.*;
import org.swt.edu.jlego.*;

public final class WheelComponent implements Wheel {

 private org.swt.edu.jlego.example.wheel.tire.Tire tire;
 private org.swt.edu.jlego.example.wheel.rim.Rim rim;

 public static Wheel create(String rim_brand, int size) throws
ConditionException {
  return new WheelComponent(rim_brand, size);
 }

 private WheelComponent(String rim_brand, int size) throws ConditionException
{
  try {
   if(!((tire == null) && (rim == null))){
    throw new PreConditionException("modules were previously initialized");
   }

   this.tire = TireModule.create("Michelin", size);
   this.rim = RimModule.create(rim_brand, size);

   if(!((tire != null) && (rim != null))){
    throw new PostConditionException("modules weren't initialized");
   }
   if(!(rim.getBrand().equals(rim_brand))){
    throw new PostConditionException("brand wasn't initialized");
   }
  }
  catch(ConditionException e) {
   throw e;
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
 }

 protected void finalize(){
  try{
   destroy();
  }
```

```
 catch(Throwable t){}
}

public void destroy() throws ConditionException {
 try {
  this.tire = null;
  this.rim = null;
 }
 catch(Exception e) {
  throw new ConditionException(e.getMessage());
 }
}

public int getPressure(int temperature) throws ConditionException {
 int result = 0;
 boolean has_ex_occured = false;

 try {
  if(!((temperature >= 0) && (temperature <= 200))){
   throw new PreConditionException("invalid temperature");
  }

  result = tire().getPressure(temperature);
  return result;
 }
 catch(ConditionException e) {
  has_ex_occured = true;
  throw e;
 }
 catch(Exception e) {
  has_ex_occured = true;
  throw new ConditionException(e.getMessage());
 }
 finally {
  try {
   if(!has_ex_occured) {
    if(!((result >= 0) && (result <= 200))){
     throw new PostConditionException("invalid pressure");
    }
   }
  }
  catch(ConditionException e) {
   throw e;
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
```

```
    }
   }
}

public boolean setPressure(int pressure) throws ConditionException {
 try {
  if(!((pressure >= 0) && (pressure <= 100))){
   throw new PreConditionException("invalid pressure");
  }

  return tire().setPressure(pressure);
 }
 catch(ConditionException e) {
  throw e;
 }
 catch(Exception e) {
  throw new ConditionException(e.getMessage());
 }
}

public int getSize() throws ConditionException {
 int result = 0;
 boolean has_ex_occured = false;

 try {
  result = rim().getSize();
  return result;
 }
 catch(Exception e) {
  has_ex_occured = true;
  throw new ConditionException(e.getMessage());
 }
 finally {
  try {
   if(!has_ex_occured) {
    if(!((result >= 0) && (result <= 30))){
     throw new PostConditionException("invalid size");
    }
   }
  }
  catch(ConditionException e) {
   throw e;
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
```

```
  }
}

public int getRotationalAngle() throws ConditionException {
 int result = 0;
 boolean has_ex_occured = false;

 try {
  result = rim().getRotationalAngle();
  return result;
 }
 catch(Exception e) {
  has_ex_occured = true;
  throw new ConditionException(e.getMessage());
 }
 finally {
  try {
   if(!has_ex_occured) {
    if(!(result >= 0 && result <= 360)){
     throw new PostConditionException("invalid angle");
    }
   }
  }
  catch(ConditionException e) {
   throw e;
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
 }
}

private void checkInvariants() {
 if(!(tire.getSize() == rim.getSize())){
  throw new InvariantException("wheel size is inconsistent");
 }

 if(!((tire != null) && (rim != null))){
  throw new InvariantException("rim or tire is null");
 }
}

private org.swt.edu.jlego.example.wheel.tire.Tire tire() {
 checkInvariants();

 return tire;
```

```
  }

  private org.swt.edu.jlego.example.wheel.rim.Rim rim() {
   checkInvariants();

   return rim;
  }

}


package org.swt.edu.jlego.example.truckwheel;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.example.truckwheel.sporttire.*;
import org.swt.edu.jlego.example.wheel.*;
import org.swt.edu.jlego.*;

public interface TruckWheel {

  public interface SportTire extends
org.swt.edu.jlego.example.truckwheel.sporttire.SportTire {}

  public void destroy() throws ConditionException;

  public int getPressure(int temperature) throws ConditionException;

  public boolean setPressure(int pressure) throws ConditionException;

  public int getSize() throws ConditionException;

  public int getRotationalAngle() throws ConditionException;

}


package org.swt.edu.jlego.example.truckwheel;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.example.truckwheel.sporttire.*;
import org.swt.edu.jlego.example.wheel.*;
import org.swt.edu.jlego.*;

public final class TruckWheelComponent implements TruckWheel {

  private org.swt.edu.jlego.example.wheel.Wheel wheel;
  private org.swt.edu.jlego.example.truckwheel.sporttire.SportTire tire;
```

```
 public static TruckWheel create(String rim_brand, int size) throws Exception,
ConditionException {
  return new TruckWheelComponent(rim_brand, size);
 }

 private TruckWheelComponent(String rim_brand, int size) throws Exception,
ConditionException {
  try {
   this.wheel = WheelComponent.create(rim_brand, size);
   this.tire = SportTireModule.create();
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
 }

 protected void finalize(){
  try{
   destroy();
  }
  catch(Throwable t){}
 }

 public void destroy() throws ConditionException {
  try {
   this.wheel = null;
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
 }

 public int getPressure(int temperature) throws ConditionException {
  try {
   return tire().getPressure(temperature);
  }
  catch(Exception e) {
   throw new ConditionException(e.getMessage());
  }
 }

 public boolean setPressure(int pressure) throws ConditionException {
  try {
   if(!((pressure >= 0) && (pressure <= 30))){
    throw new PreConditionException("invalid pressure");
```

```
    }

    return tire().setPressure(pressure);
   }
   catch(ConditionException e) {
    throw e;
   }
   catch(Exception e) {
    throw new ConditionException(e.getMessage());
   }
  }

  public int getSize() throws ConditionException {
   try {
    return wheel().getSize();
   }
   catch(Exception e) {
    throw new ConditionException(e.getMessage());
   }
  }

  public int getRotationalAngle() throws ConditionException {
   try {
    return wheel().getRotationalAngle();
   }
   catch(Exception e) {
    throw new ConditionException(e.getMessage());
   }
  }

  private void checkInvariants() {
   if(!(wheel != null)){
    throw new InvariantException("wheel is null");
   }

   if(!(tire != null)){
    throw new InvariantException("tire is null");
   }
  }

  private org.swt.edu.jlego.example.wheel.Wheel wheel() {
   checkInvariants();

   return wheel;
  }
```

```
  private org.swt.edu.jlego.example.truckwheel.sporttire.SportTire tire() {
    checkInvariants();

    return tire;
  }

}


package org.swt.edu.jlego.example.wheel.rim;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public interface Rim {

  public int getRotationalAngle();

  public int getSize();

  public String getBrand();

}


package org.swt.edu.jlego.example.wheel.rim;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public final class RimModule implements Rim {

  private org.swt.edu.jlego.example.wheel.rim.CarRim rim;

  public static Rim create(String brand, int size) {
    return new RimModule(brand, size);
  }

  private RimModule(String brand, int size) {
    rim = new CarRim(brand, size);
  }

  protected void finalize(){
    try{
      destroy();
    }
```

```
    catch(Throwable t){}
  }

  public void destroy() {
  }

  public int getRotationalAngle() {
    return rim.getRotationalAngle();
  }

  public int getSize() {
    return rim.getSize();
  }

  public String getBrand() {
    return rim.getBrand();
  }

}


package org.swt.edu.jlego.example.truckwheel.sporttire;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public interface SportTire {

  public int getPressure(int temperature);

  public boolean setPressure(int pressure);

}


package org.swt.edu.jlego.example.truckwheel.sporttire;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public final class SportTireModule implements SportTire {

  private org.swt.edu.jlego.example.truckwheel.sporttire.PressureSensor sensor;

  public static SportTire create() {
    return new SportTireModule();
```

```
  }

  private SportTireModule() {
    sensor = new PressureSensor(28);
  }

  protected void finalize(){
    try{
      destroy();
    }
    catch(Throwable t){}
  }

  public void destroy() {
    sensor.stop();
  }

  public int getPressure(int temperature) {
    return sensor.getPressure(temperature);
  }

  public boolean setPressure(int pressure) {
    return sensor.setPressure(pressure);
  }

}


package org.swt.edu.jlego.example.wheel.tire;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public interface Tire {

  public int getPressure(int temperature);

  public boolean setPressure(int pressure) throws
UnsupportedOperationException;

  public int getSize();

  public String getBrand();

}
```

```
package org.swt.edu.jlego.example.wheel.tire;

import org.swt.edu.jlego.util.*;
import org.swt.edu.jlego.*;

public final class TireModule implements Tire {

 private org.swt.edu.jlego.example.wheel.tire.CarTire tire;

 public static Tire create(String brand, int size) {
  return new TireModule(brand, size);
 }

 private TireModule(String brand, int size) {
  tire = new CarTire(brand, size);
 }

 protected void finalize(){
  try{
   destroy();
  }
  catch(Throwable t){}
 }

 public void destroy() {
 }

 public int getPressure(int temperature) {
  throw new UnsupportedOperationException("generic tire doesn't support
pressure reading");
 }

 public boolean setPressure(int pressure) throws UnsupportedOperationException
{
  throw new UnsupportedOperationException("generic tire doesn't support self-
inflation");
 }

 public int getSize() {
  return tire.getSize();
 }

 public String getBrand() {
  return tire.getBrand();
 }
```

```
}
```

Appendix G

```
package org.swt.edu.jlego;

public class ConditionException extends Exception {

 public ConditionException(String message) {
  super(message);
 }

}


package org.swt.edu.jlego;

public class InvariantException extends RuntimeException {

 public InvariantException(String message) {
  super(message);
 }

}


package org.swt.edu.jlego;

public class JLegoError extends Error {

 public JLegoError(String message) {
  super(message);
 }

}


package org.swt.edu.jlego;

public class PostConditionException extends ConditionException {

 public PostConditionException(String message) {
  super(message);
```

```
  }

}


package org.swt.edu.jlego;

public class PreConditionException extends ConditionException {

  public PreConditionException(String message) {
    super(message);
  }

}


package org.swt.edu.jlego.util;

import java.io.*;
import java.util.*;
import java.util.jar.*;
import java.util.zip.*;

public class Files {
  public static final String MODULE_EXT = ".module";
  public static final String COMPONENT_EXT = ".component";
  public static final String CLASS_EXT = ".class";
  public static final String JAVA_EXT = ".java";
  public static final String COMPONENT_ARCHIVE_EXT = ".car";
  public static final String MODULE_ARCHIVE_EXT = ".mar";

  private static final byte[] buffer = new byte[8096];

  public static boolean isArchive(File file) {
    return (file.getName().endsWith(COMPONENT_ARCHIVE_EXT) ||
file.getName().endsWith(MODULE_ARCHIVE_EXT));
  }

  public static boolean isImportable(File file) {
    return (isCompilable(file.getName()) ||
(file.getName().endsWith(CLASS_EXT)) ||
(file.getName().endsWith(JAVA_EXT)));
  }

  public static boolean isCompilable(File file) {
    return isCompilable(file.getName());
```

```
  }

  public static boolean isCompilable(String name) {
    return (name.endsWith(MODULE_EXT) ||
name.endsWith(COMPONENT_EXT));
  }

  public static boolean isModule(String name) {
    return name.endsWith(MODULE_EXT);
  }

  public static boolean isComponent(String name) {
    return name.endsWith(COMPONENT_EXT);
  }

  public static boolean isJava(String name) {
    return name.endsWith(JAVA_EXT);
  }

  public static void archiveDirectory(JarOutputStream out, File root, File dir)
throws IOException {
    File files[] = dir.listFiles();
    for (int i = 0; i < files.length; i++) {
      if (files[i].isDirectory()) {
        archiveDirectory(out, root, files[i]);
      }
      else {
        JarEntry entry = new
JarEntry(files[i].getPath().substring(root.getPath().length() +
1).replace(File.separatorChar, '/'));
        out.putNextEntry(entry);
        InputStream in = new FileInputStream(files[i]);
        copy(in, out, false);
        out.closeEntry();
        in.close();
      }
    }
  }

  public static List getFileListing(File root, File dir) throws IOException {
    return getFileListing(new ArrayList(), root, dir);
  }

  public static List getFileListing(List list, File root, File dir) throws IOException
{
    File files[] = dir.listFiles();
```

```
    for (int i = 0; i < files.length; i++) {
      if (files[i].isDirectory()) {
        getFileListing(list, root, files[i]);
      }
      else {
        if (isArchive(files[i])) {
          ZipFile zip = new ZipFile(files[i]);
          Enumeration enum = zip.entries();
          while (enum.hasMoreElements()) {
            list.add(((ZipEntry) enum.nextElement()).getName());
          }
        }

        list.add(files[i].getPath().substring(root.getPath().length() + 1));
      }
    }

    return list;
  }

  public static void copy(File archive, File to) throws IOException {
    ZipFile zip = new ZipFile(archive);
    Enumeration enum = zip.entries();

    while (enum.hasMoreElements()) {
      ZipEntry entry = (ZipEntry) enum.nextElement();

      File file = new File(to, entry.getName());

      if (file.getPath().indexOf("META-INF") != -1) {
        continue;
      }

      new File(to, entry.getName().substring(0, entry.getName().length() -
file.getName().length())).mkdirs();

      copy(zip.getInputStream(entry), new FileOutputStream(file), true);
    }
  }

  public static void copy(File from, List list, File to) throws IOException {
    for (Iterator iter = list.iterator(); iter.hasNext(); ) {
      String item = (String) iter.next();
      File file = new File(from, item);

      if (file.getPath().indexOf("META-INF") != -1) {
```

```
      continue;
     }

     if (file.isFile()) {
      new File(to, item.substring(0, item.length() -
file.getName().length())).mkdirs();
     }

     copy(new FileInputStream(file), new FileOutputStream(new File(to, item)),
true);
    }
  }

  public static synchronized long copy(InputStream in, OutputStream out, boolean
close) throws IOException {
    long bytes_read = 0;
    for (int read = in.read(buffer); read != -1; read = in.read(buffer)) {
     out.write(buffer, 0, read);
     bytes_read += read;
    }

    if (close) {
     in.close();
     out.close();
    }

    return bytes_read;
  }

  public static void deleteDirectory(File dir, boolean remove, boolean
delete_archives) {
    File files[] = dir.listFiles();
    for (int i = 0; i < files.length; i++) {
     if (files[i].isDirectory()) {
      deleteDirectory(files[i], true, delete_archives);
     }
     else {
      if (!delete_archives && isArchive(files[i])) {
       continue;
      }

      files[i].delete();
     }
    }

    if (remove) {
```

```
    dir.delete();
   }
  }

  public static File getPath(String path, boolean create) throws IOException {
   try {
    File file = new File(path);

    if(file.exists()){
     if (!file.isDirectory()) {
      throw new IOException("not a directory: " + path);
     }

     if (!file.canWrite() || !file.canRead()) {
      throw new IOException("no read/write permission: " + path);
     }
    }
    else{
     if(!file.mkdirs()){
      throw new IOException("unable to create: " + path);
     }
    }

    return file;
   }
   catch (IOException e) {
    throw e;
   }
  }

}


package org.swt.edu.jlego.util;

import java.io.*;
import java.util.*;
import java.util.zip.*;

import org.swt.edu.jlego.*;

public final class ModuleFile {
 private HashMap contents;

 public ModuleFile(String name) {
  this.contents = new HashMap(20, .9f);
```

```
    name = name + ".mar";

    try {
      InputStream in =
ClassLoader.getSystemClassLoader().getResourceAsStream(name);

      if (in == null) {
        throw new JLegoError("module is not in the search path -> " + name);
      }

      load(in);

      if (isEmpty()) {
        throw new JLegoError("module is empty -> " + name);
      }
    }
    catch (Exception e) {
      e.printStackTrace();
      throw new JLegoError("error while loading module -> " + name);
    }
  }

  public boolean isEmpty() {
    return contents.isEmpty();
  }

  public Set getContent() {
    return contents.keySet();
  }

  public byte[] getClass(String name) {
    Resource resource = ((Resource) contents.get(name + ".class"));
    if (resource != null) {
      return resource.getData();
    }

    return null;
  }

  public boolean containsClass(String name) {
    return contents.containsKey(name + ".class");
  }

  public long getSize(String name) {
    Resource resource = ((Resource) contents.get(name));
```

```
    if (resource != null) {
      return resource.getSize();
    }

    return -1;
  }

  public Properties getProperties(String name) {
    if (contents.containsKey(name)) {
      ByteArrayInputStream in = new ByteArrayInputStream(((Resource)
contents.get(name)).getData());
      Properties props = new Properties();

      try {
        props.load(in);
      }
      catch (Exception e) { }

      return props;
    }

    return null;
  }

  public void unload() {
    contents.clear();
  }

  public void load(String file) throws IOException {
    load(new FileInputStream(file));
  }

  public void load(InputStream in) throws IOException {
    ZipInputStream zis = null;

    try {
      zis = new ZipInputStream(new BufferedInputStream(in, 1024));

      ZipEntry ze = null;
      while ((ze = zis.getNextEntry()) != null) {
        if (ze.isDirectory()) {
          continue;
        }

        int size = (int) ze.getSize();
        byte[] buffer = new byte[(int) size];
```

```
      int remaining = 0;
      int chunk = 0;
      while ((size - remaining) > 0) {
       chunk = zis.read(buffer, remaining, size - remaining);
       if (chunk == -1) {
        break;
       }

       remaining += chunk;
      }

      contents.put(ze.getName(), new Resource(ze.getName(), buffer));
     }
    }
    finally {
     try {
      zis.close();
     }
     catch (Exception ex) {}
    }
  }

  public String toString() {
    StringBuffer buffer = new StringBuffer(512);
    Set set = getContent();
    Iterator iter = set.iterator();
    while (iter.hasNext()) {
     buffer.append((String) iter.next()).append("\n");
    }

    return buffer.toString();
  }

  private class Resource {
    private long size;
    private byte[] data;
    private String name;

    public Resource(String name, byte[] data) {
     this(name, data.length);

     this.data = data;
    }

    public Resource(String name, long size) {
     this.size = size;
```

```
    this.name = name;
   }

   public long getSize() {
    return size;
   }

   public String getName() {
    return name;
   }

   public void setData(byte[] data) {
    this.data = data;
    this.size = data.length;
   }

   public byte[] getData() {
    return data;
   }
  }
}


package org.swt.edu.jlego.util;

import java.lang.reflect.*;
import java.util.*;

public class ModuleLoader extends ClassLoader {
 private static List modules = new ArrayList();
 private static Map constructors = new HashMap();
 private static ModuleLoader loader = new ModuleLoader();

 private Map classes = new HashMap();

 public static Object loadModule(String name, Object params[]) {
  try {
   if (constructors.containsKey(name)) {
    return ((Constructor) constructors.get(name)).newInstance(params);
   }

   ModuleFile module = new ModuleFile(name);
   modules.add(module);

   Properties props = module.getProperties("META-INF/MODULE.INF");
   Class c = loader.loadClass(props.getProperty("module-class"));
```

```
    Class types[] = new Class[params.length];
    for (int i = 0; i < params.length; i++) {
      types[i] = params[i].getClass();
    }

    Constructor con = c.getConstructor(types);
    constructors.put(name, con);

    return con.newInstance(params);
   }
   catch (Exception e) {
    e.printStackTrace();
   }

   return null;
  }

  public Class findClass(String name) throws ClassNotFoundException {
   name = name.replace('.', '/');

   if (classes.containsKey(name)) {
    return (Class) classes.get(name);
   }

   ModuleFile module = getModule(name);

   if (module != null) {
    byte[] bytes = module.getClass(name);

    if (bytes != null) {
     try {
      Class c = defineClass(name, bytes, 0, bytes.length);
      classes.put(name, c);

      return c;
     }
     catch (Throwable t) {
      t.printStackTrace();
     }
    }
   }

   throw new ClassNotFoundException(name);
  }
```

```
  private ModuleFile getModule(String name) {
    for (int i = 0; i < modules.size(); i++) {
      ModuleFile module = (ModuleFile) modules.get(i);

      if (module.containsClass(name)) {
        return module;
      }
    }

    return null;
  }
}


package org.swt.edu.jlego.compiler;

import java.io.*;
import java.util.*;

import org.swt.edu.jlego.model.*;

public abstract class AbstractCompiler {
  protected File file;
  protected File build_dir;
  protected File source_dir;
  protected CompileError error;
  private HashMap resolved_vars;

  public AbstractCompiler() {
    this.resolved_vars = new HashMap(15);
  }

  public abstract Unit getUnit();

  protected abstract boolean execute();

  public void setFile(File file) {
    this.file = file;
  }

  public void setBuildDirectory(File dir) {
    this.build_dir = dir;
  }

  public void setSourceDirectory(File dir) {
    this.source_dir = dir;
```

```
  }

  public CompileError getError() {
    return error;
  }

  public boolean hasError() {
    return (error != null);
  }

  public boolean compile() {
    resolved_vars.clear();
    error = null;

    if (build_dir == null) {
      throw new CompileError(getCompilerName(), "please specify a build
directory");
    }

    if (file == null) {
      throw new CompileError(getCompilerName(), "please specify a component
file");
    }

    return execute();
  }

  public String getCompilerName() {
    return ((this instanceof ComponentCompiler) ? "ComponentCompiler" :
"ModuleCompiler");
  }

  protected void modify(Unit unit) {
    unit.addImport(new Import("org.swt.edu.jlego.*"));
    unit.addImport(new Import("org.swt.edu.jlego.util.*"));
  }

  protected void verifyName(Unit unit) {
    if (!unit.getName().equals(file.getName().substring(0, Math.max(0,
file.getName().indexOf('.'))))) {
      String pkg = unit.hasPackage() ? unit.getPackage().getPath() + '/' : "";
      throw new CompileError(getCompilerName(), unit.getName() + " shouldn't
be in " + pkg + file.getName());
    }
  }
```

```
  protected void verifyPackage(Unit unit) {
    String file_path = file.getPath();

    int index = file_path.lastIndexOf(File.separatorChar);
    String package_path =
file_path.substring(Math.min(source_dir.getPath().length() + 1, index),
index).replace(File.separatorChar, '.');

    if (!unit.hasPackage() && package_path.length() > 0) {
      throw new CompileError(getCompilerName(), unit.getName() + " should be
in package " + unit.getName().toLowerCase());
    }
    else if (!package_path.equals(unit.getPackage().getName())) {
      throw new CompileError(getCompilerName(),
                    package_path.replace('.', '/') + '/' + unit.getName() + " should be
in package " + unit.getPackage().getPath());
    }
    else if (!package_path.endsWith(unit.getName().toLowerCase())) {
      throw new CompileError(getCompilerName(),
                    package_path.replace('.', '/') + '/' + unit.getName() + " should be
in package *." + unit.getName().toLowerCase());
    }
  }

  protected File createSourcePath(Unit unit, String name) throws IOException {
    return createPath(unit, source_dir.getAbsolutePath() + File.separatorChar,
name);
  }

  protected File createBuildPath(Unit unit, String name) throws IOException {
    return createPath(unit, build_dir.getAbsolutePath() + File.separatorChar,
name);
  }

  protected File createPath(Unit unit, String path, String name) throws
IOException {
    if (unit.hasPackage()) {
      path += unit.getPackage().getPath() + File.separatorChar;
    }

    File file = new File(path);
    if (!file.exists() && !file.mkdirs()) {
      throw new CompileError(getCompilerName(), "unable to create path: " +
path);
    }
```

```
    path += name;

    path += ".java";

    return new File(path);
  }

  protected void writeInterface(Unit unit) throws IOException {
    PrintWriter out = new PrintWriter(new
FileOutputStream(createSourcePath(unit, unit.getName())));

    if (unit.hasPackage()) {
      unit.getPackage().write(out);
      out.println();
    }

    if (unit.hasImports()) {
      unit.getImports().write(out);
      out.println();
    }

    writeInterfaceDeclaration(out, unit);

    writeInterfaceHeader(out, unit);

    Methods methods = unit.getMethods();

    for (int i = 0; i < methods.getSize(); i++) {
      Method method = methods.getMethod(i);

      out.print("  public ");
      out.print(method.getReturnType());
      out.print(" ");
      out.print(method.getName());
      out.print("(");

      if (method.hasParameters()) {
        Parameters params = method.getParameters();
        params.write(out);
      }

      out.print(")");

      if (method.hasExceptions()) {
        Exceptions exps = method.getExceptions();
```

```
      exps.write(out);
    }

    out.println(";");
    out.println();
  }

  writeInterfaceFooter(out, unit);

  out.println("}");

  out.flush();
  out.close();
}

protected void writeInterfaceDeclaration(PrintWriter out, Unit unit) {
  try {
    out.print("public interface ");
    out.print(unit.getName());

    if (unit.hasComposition()) {
      out.print(" implements ");

      Composition comp = unit.getComposition();
      for (int i = 0; i < comp.getSize(); i++) {
        Composite c = comp.get(i);

        if (unit instanceof Component) {
          ClassPath.instance().findComponent(unit, c.getName());
        }
        else {
          ClassPath.instance().findModule(unit, c.getName());
        }

        out.print(c.getName());

        if ((i + 1) < comp.getSize()) {
          out.print(", ");
        }
      }
    }

    out.println(" {");
    out.println();
  }
  catch (ModuleNotFoundException e) {
```

```
    throw new CompileError("ModuleCompiler", e.getMessage());
   }
   catch (ComponentNotFoundException e) {
    throw new CompileError("ComponentCompiler", e.getMessage());
   }
  }

  protected void writeInitializer(PrintWriter out, Unit unit, String name) {
   writeInitializer(out, unit, name, null);
  }

  protected void writeInitializer(PrintWriter out, Unit unit, String name,
Constructor c) {
   out.print("  public static ");
   out.print(unit.getName());
   out.print(" create(");

   if ((c != null) && c.hasParameters()) {
    c.getParameters().write(out);
   }

   out.print(")");

   if ((c != null) && c.hasExceptions()) {
    c.getExceptions().write(out);
   }

   out.println(" {");

   out.print("    return new ");
   out.print(name);
   out.print("(");

   if ((c != null) && c.hasParameters()) {
    c.getParameters().writeVariables(out);
   }

   out.println(");");

   out.println("  }");
   out.println();
  }

  protected void writeConstructor(PrintWriter out, Unit unit, String name) {
   writeConstructor(out, unit, name, null);
  }
```

```
 protected void writeConstructor(PrintWriter out, Unit unit, String name,
Constructor c) {
  out.print("  private ");
  out.print(name);
  out.print("(");

  if ((c != null) && (c.hasParameters())) {
   c.getParameters().write(out);
  }

  out.print(")");

  if (c.hasExceptions()) {
   c.getExceptions().write(out);
  }

  out.println(" {");

  if (c != null) {
   if (c.getOperation().hasConstructorInvocation()) {
    /** @todo check if invocation is valid */

    out.print("    ");
    c.getOperation().getConstructorInvocation().write(out);
    out.println();
   }

   writeConstructorBody(out, c);
  }

  out.println("  }");
  out.println();
 }

 protected void writeConstructorBody(PrintWriter out, Constructor c) {
  writeOperation(out, c.getOperation());
 }

 protected void writeDestructor(PrintWriter out, Destructor d) {
  out.println("  protected void finalize(){");
  out.println("    try{");
  out.println("      destroy();");
  out.println("    }");
  out.println("    catch(Throwable t){}");
  out.println("  }\n");
```

```
  out.print("  public void destroy()");

  if (d.hasExceptions()) {
   d.getExceptions().write(out);
  }

  out.println(" {");

  writeDestructorBody(out, d);

  out.println("  }");
  out.println();
 }

 protected void writeDestructorBody(PrintWriter out, Destructor d) {
  writeOperation(out, d.getOperation());
 }

 protected void writeOperation(PrintWriter out, Operation op) {
  writeOperation(out, op, "    ");
 }

 protected void writeOperation(PrintWriter out, Operation op, String space) {
  if (op.hasStatements()) {
    Statements statements = op.getStatements();

    for (int i = 0; i < statements.getSize(); i++) {
     Statement statement = statements.get(i);

     statement.convert(this);
     out.print(space);
     statement.write(out);
    }
  }
 }

 protected void writeVariable(PrintWriter out, Unit unit, Variable v) {
  out.print("  private ");
  out.print(resolveVariable(unit, v, true));
  out.print(" ");
  out.print(v.getName());
  out.println(";");
 }

 protected String resolveVariable(Unit unit, Variable v, boolean full) {
```

```
    try {
      String absolute = null;

      if (!resolved_vars.containsKey(v.getName())) {
        if ("module".equals(v.getMeta())) {
          absolute = ClassPath.instance().findModule(unit, v.getType());
        }
        else if ("component".equals(v.getMeta())) {
          absolute = ClassPath.instance().findComponent(unit, v.getType());
        }
        else if ("class".equals(v.getMeta())) {
          absolute = ClassPath.instance().findClass(unit, v.getType());
        }
        else {
          throw new java.lang.Exception("unknown variable meta type: " +
v.getMeta());
        }

        v.setAbsoluteType(absolute);
        resolved_vars.put(v.getName(), v);
      }
      else {
        absolute = ((Variable) resolved_vars.get(v.getName())).getAbsoluteType();
      }

      if (full) {
        return absolute;
      }
      else {
        return v.getType();
      }
    }
    catch (java.lang.Exception ex) {
      throw new CompileError(getCompilerName(), ex.getMessage());
    }
  }

  protected void writeMethod(PrintWriter out, Method method) {
    out.print("  public ");
    out.print(method.getReturnType());
    out.print(" ");
    out.print(method.getName());
    out.print("(");

    if (method.hasParameters()) {
      method.getParameters().write(out);
```

```
    }

    out.print(")");

    if (method.hasExceptions()) {
      method.getExceptions().write(out);
    }

    out.println(" {");

    writeMethodBody(out, method);

    out.println("  }");
    out.println();
  }

  protected void writeMethodBody(PrintWriter out, Method method) {
    writeOperation(out, method.getOperation());
  }

  protected void writeUnit(Unit unit) throws IOException {
    String name = unit.getName() + ((unit instanceof Module) ? "Module" :
"Component");
    PrintWriter out = new PrintWriter(new
FileOutputStream(createSourcePath(unit, name)));

    if (unit.hasPackage()) {
      unit.getPackage().write(out);
      out.println();
    }

    if (unit.hasImports()) {
      unit.getImports().write(out);
      out.println();
    }

    out.print("public final class ");
    out.print(name);
    out.print(" implements ");
    out.print(unit.getName());
    out.println(" {");
    out.println();

    writeUnitHeader(out, unit);

    if (unit.hasVariables()) {
```

```
    Variables vars = unit.getVariables();

    for (int i = 0; i < vars.getSize(); i++) {
      writeVariable(out, unit, vars.getVariable(i));
    }

    out.println();
  }

  if (unit.hasConstructors()) {
    Constructors consts = unit.getConstructors();

    for (int i = 0; i < consts.getSize(); i++) {
      writeInitializer(out, unit, name, consts.getConstructor(i));
    }

    for (int i = 0; i < consts.getSize(); i++) {
      writeConstructor(out, unit, name, consts.getConstructor(i));
    }
  }
  else {
    writeInitializer(out, unit, name);
    writeConstructor(out, unit, name);
  }

  if (unit.hasDestructor()) {
    writeDestructor(out, unit.getDestructor());
  }

  Methods methods = unit.getMethods();

  for (int i = 0; i < methods.getSize(); i++) {
    Method method = methods.getMethod(i);

    writeMethod(out, method);
  }

  writeUnitFooter(out, unit);

  out.println("}");

  out.flush();
  out.close();
}

protected void writeUnitHeader(PrintWriter out, Unit unit) { }
```

```
  protected void writeUnitFooter(PrintWriter out, Unit unit) {}

  protected void writeInterfaceHeader(PrintWriter out, Unit unit) {}

  protected void writeInterfaceFooter(PrintWriter out, Unit unit) {}

  protected String getInitializerForType(String type) {
    if (type.equals("int") || type.equals("float") || type.equals("long") ||
type.equals("double") || type.equals("byte") || type.equals("short") ||
        type.equals("char")) {
      return "0";
    }

    return "null";
  }

}


package org.swt.edu.jlego.compiler;

import java.io.*;
import java.util.*;

import org.swt.edu.jlego.model.*;
import org.swt.edu.jlego.util.*;

public class ClassPath {
  private static ClassPath self = new ClassPath();

  public static ClassPath instance() {
    return self;
  }

  private List paths;
  private HashMap cached_paths;
  private HashSet cache;

  public ClassPath() {
    this.cache = new HashSet(50, .9f);
    this.paths = new ArrayList(10);
    this.cached_paths = new HashMap(50, .9f);

    StringTokenizer tokenizer = new
StringTokenizer(System.getProperty("java.class.path"), File.pathSeparator);
```

```
    while (tokenizer.hasMoreTokens()) {
      String path = tokenizer.nextToken();

      try {
        File file = new File(normalize(path));
        if (!file.canRead()) {
          throw new IOException("classpath element is unreadable");
        }

        this.paths.add(file);
      }
      catch (IOException e) {
        System.err.println(e.getMessage() + ": " + path);
      }
    }
  }

  public String findClass(Unit unit, String name) throws ClassNotFoundException
{
    String c = find(unit, name, Files.JAVA_EXT);
    if (c != null) {
      return c.substring(0, c.length() -
Files.JAVA_EXT.length()).replace(File.separatorChar, '.');
    }

    c = find(unit, name, Files.CLASS_EXT);
    if (c != null) {
      return c.substring(0, c.length() -
Files.CLASS_EXT.length()).replace(File.separatorChar, '.');
    }

    throw new ClassNotFoundException("unable to locate the class: " + name);
  }

  public String findModule(Unit unit, String name) throws
ModuleNotFoundException {
    String module = find(unit, name, Files.MODULE_EXT);
    if (module != null) {
      return module.substring(0, module.length() -
Files.MODULE_EXT.length()).replace(File.separatorChar, '.');
    }

    module = find(unit, name + "Module", Files.JAVA_EXT);
    if (module != null) {
```

```
    return module.substring(0, module.length() - "Module".length() -
Files.JAVA_EXT.length()).replace(File.separatorChar, '.');
   }

   module = find(unit, name + "Module", Files.CLASS_EXT);
   if (module != null) {
    return module.substring(0, module.length() - "Module".length() -
Files.CLASS_EXT.length()).replace(File.separatorChar, '.');
   }

   module = find(unit, name, Files.MODULE_ARCHIVE_EXT);
   if (module != null) {
    return module.substring(0, module.length() -
Files.MODULE_ARCHIVE_EXT.length()).replace(File.separatorChar, '.');
   }

   throw new ModuleNotFoundException("unable to locate the module: " +
name);
  }

  public String findComponent(Unit unit, String name) throws
ComponentNotFoundException {
   String component = find(unit, name, Files.COMPONENT_EXT);
   if (component != null) {
    return component.substring(0, component.length() -
Files.COMPONENT_EXT.length()).replace(File.separatorChar, '.');
   }

   component = find(unit, name + "Component", Files.JAVA_EXT);
   if (component != null) {
    return component.substring(0, component.length() - "Component".length() -
Files.JAVA_EXT.length()).replace(File.separatorChar, '.');
   }

   component = find(unit, name + "Component", Files.CLASS_EXT);
   if (component != null) {
    return component.substring(0, component.length() - "Component".length() -
Files.CLASS_EXT.length()).replace(File.separatorChar, '.');
   }

   component = find(unit, name, Files.COMPONENT_ARCHIVE_EXT);
   if (component != null) {
    return component.substring(0, component.length() -
Files.COMPONENT_ARCHIVE_EXT.length()).replace(File.separatorChar, '.');
   }
```

```
    throw new ComponentNotFoundException("unable to locate the component: "
+ name);
  }

  public String findModulePath(Unit unit, String name) throws
ModuleNotFoundException {
    String module = find(unit, name, Files.MODULE_ARCHIVE_EXT);

    if ((module != null) && cached_paths.containsKey(module)) {
      return (String) cached_paths.get(module);
    }

    throw new ModuleNotFoundException("unable to locate the module path: " +
name);
  }

  public String findComponentPath(Unit unit, String name) throws
ComponentNotFoundException {
    String comp = find(unit, name, Files.COMPONENT_ARCHIVE_EXT);

    if ((comp != null) && cached_paths.containsKey(comp)) {
      return (String) cached_paths.get(comp);
    }

    throw new ComponentNotFoundException("unable to locate the component
path: " + name);
  }

  private String find(Unit unit, String name, String ext) {
    List names = new ArrayList(5);

    names.add(unit.hasPackage() ? normalize(unit.getPackage().getPath()) +
File.separator + name + ext : name);

    if (unit.hasImports()) {
      Imports imports = unit.getImports();

      for (Iterator iter = imports.iterator(); iter.hasNext(); ) {
        Import imp = (Import) iter.next();

        if (imp.isGlobal()) {
          names.add(normalize(imp.getPath().replace('.', File.separatorChar)) +
File.separator + name + ext);
        }
        else {
          names.add(normalize(imp.getPath().replace('.', File.separatorChar)) + ext);
```

```
      }
    }
  }

  for (int i = 0; i < names.size(); i++) {
    if (cache.contains(names.get(i))) {
      return (String) names.get(i);
    }
  }

  for (int i = 0; i < paths.size(); i++) {
    String full_name = findInPath(names, (File) paths.get(i));

    if (full_name != null) {
      full_name = full_name;
      cache.add(full_name);

      return full_name;
    }
  }

  return null;
}

private String findInPath(List names, File file) {
  for (int j = 0; j < names.size(); j++) {
    String path = file.toString() + File.separatorChar + (String) names.get(j);
    File match = new File(path);

    if (!match.exists() || !match.canRead()) {
      continue;
    }

    if (Files.isArchive(match)) {
      cached_paths.put(names.get(j), match.toString());
    }

    return (String) names.get(j);
  }

  return null;
}

private String normalize(String path) {
  path = path.replace('/', File.separatorChar);
  path = path.replace('\\', File.separatorChar);
```

```
    path = path.replace(':', File.separatorChar);

    return path;
  }

  public static void main(String args[]) throws Throwable {
    System.setProperty("java.class.path", "./test/src;test.jar");

    Unit unit = new Module();
    unit.addImport(new Import("org.swt.edu.jlego.example.wheel.rim.*"));
    unit.addImport(new Import("org.swt.edu.jlego.example.wheel.tire.*"));

    System.err.println("found: " + ClassPath.instance().findModule(unit, "Tire"));
    System.err.println("found: " + ClassPath.instance().findModule(unit, "Tire"));
  }

}


package org.swt.edu.jlego.compiler;

public class CompileError extends Error {

  public CompileError(String type, String message) {
    super("[" + type + "] " + message);
  }

  public CompileError(String type, String message, Throwable t) {
    super("[" + type + "] " + message, t);
  }

}


package org.swt.edu.jlego.compiler;

import java.io.*;
import java.util.*;
import java.util.jar.*;

import org.swt.edu.jlego.*;
import org.swt.edu.jlego.model.*;
import org.swt.edu.jlego.util.*;

public class Compiler {
  public static final String VERSION = "1.0";
```

```
private ModuleCompiler mod_compiler;
private ComponentCompiler comp_compiler;
private File build_dir;
private File source_dir;
private Set source_files;
private Map components;
private Map modules;
private List to_delete;

public Compiler(String sourcepath, String classpath, String buildpath) {
 System.setProperty("java.class.path", classpath);

 try {
  this.build_dir = Files.getPath(buildpath, true);
  this.source_dir = Files.getPath(sourcepath, false);

  this.source_files = new HashSet(20);
  this.to_delete = new ArrayList(20);
  this.components = new HashMap();
  this.modules = new HashMap();

  Files.deleteDirectory(build_dir, false, false);

  Runtime.getRuntime().addShutdownHook(
    new Thread() {
   public void run() {
    Iterator iter = to_delete.iterator();

    while (iter.hasNext()) {
     Files.deleteDirectory((File) iter.next(), true, true);
    }
   }
  }
  );

  search(source_dir);
  compile(classpath);
 }
 catch (java.lang.Exception e) {
  throw new CompileError("Compiler", "unable to initialize", e);
 }
}

private void precompile(File file) {
 if (!Files.isCompilable(file)) {
```

```
      throw new JLegoError(file.getAbsolutePath() + " isn't compilable!");
    }

    AbstractCompiler compiler = null;

    if (file.getName().endsWith(Files.MODULE_EXT)) {
      compiler = getModuleCompiler();
    }
    else {
      compiler = getComponentCompiler();
    }

    compiler.setFile(file);
    compiler.setBuildDirectory(build_dir);
    compiler.setSourceDirectory(source_dir);

    if (!compiler.compile()) {
      if (compiler.hasError()) {
        throw compiler.getError();
      }
      else {
        throw new CompileError("Compiler", "unspecified error while compiling " +
file.getAbsolutePath());
      }
    }

    Unit unit = compiler.getUnit();

    if (compiler instanceof ComponentCompiler) {
      components.put(unit.getName(), unit);
    }
    else {
      modules.put(unit.getName(), unit);
    }
  }

  private void compile(String classpath) throws IOException {
    Map mod_files = compileModules(classpath);
    Map comp_files = compileComponents(getComponentClasspath(classpath),
mod_files);
  }

  /**
   * compile modules and archive them
   * @param classpath
   * @return
```

```
   * @throws IOException
   */
  private Map compileModules(String classpath) throws IOException {
    Map mod_files = new HashMap();

    for (Iterator iter = modules.entrySet().iterator(); iter.hasNext(); ) {
      Map.Entry entry = (Map.Entry) iter.next();
      Unit unit = (Unit) entry.getValue();

      Manifest manifest = new Manifest();
      Attributes attrs = new Attributes();
      manifest.getEntries().put(unit.getName(), attrs);

      attrs.put(new Attributes.Name("Manifest-Version"), "1.0");
      attrs.put(new Attributes.Name("JLego-Type"), "module");
      attrs.put(new Attributes.Name("JLego-Version"), this.VERSION);
      attrs.put(new Attributes.Name("Interface"), (unit.hasPackage() ?
unit.getPackage().getName() + '.' : "") + unit.getName());
      attrs.put(new Attributes.Name("Implementation"), (unit.hasPackage() ?
unit.getPackage().getName() + '.' : "") + unit.getName() + "Module");

      File dir = compile(classpath, unit.getName(), unit.getPackage().getPath() +
File.separatorChar + entry.getKey() + "Module" + Files.JAVA_EXT);
      archive(unit, dir, manifest);

      mod_files.put(unit.getName(), dir);
    }

    return mod_files;
  }

  /**
   * compile components referencing the precompiled modules and components
upon which it depends
   * @param classpath
   * @param mod_files
   * @return
   * @throws IOException
   */
  private Map compileComponents(String classpath, Map mod_files) throws
IOException {
    Map comp_files = new HashMap();

    for (Iterator iter = components.entrySet().iterator(); iter.hasNext(); ) {
      Map.Entry entry = (Map.Entry) iter.next();
      Unit unit = (Unit) entry.getValue();
```

```
    /* compile */
    File dir = compile(classpath, unit.getName(), unit.getPackage().getPath() +
File.separatorChar + entry.getKey() + "Component" + Files.JAVA_EXT);

    /* archive */
    comp_files.put(unit.getName(), dir);

    if (unit.hasVariables()) {
      Variables vars = unit.getVariables();
      for (int i = 0; i < vars.getSize(); i++) {
        Variable var = vars.getVariable(i);

        if (var.getMeta().equals("module")) {
          if (mod_files.containsKey(var.getType())) {
            /* copy from module compile path */
            List list = Files.getFileListing((File) mod_files.get(var.getType()), (File)
mod_files.get(var.getType()));
            Files.copy((File) mod_files.get(var.getType()), list, dir);
          }
          else {
            try {
              /* copy from module archive */
              Files.copy(new File(ClassPath.instance().findModulePath(unit,
var.getType())), dir);
            }
            catch (ModuleNotFoundException e) {
              e.printStackTrace();
            }
          }
        }
      }
    }

    archiveComponents(comp_files);

    return comp_files;
  }

  /**
   * archives components referencing prearchived modules/components upon
which it depends
   * @param comp_files
   * @throws IOException
   */
```

```
  private void archiveComponents(Map comp_files) throws IOException {
   for (Iterator iter = components.entrySet().iterator(); iter.hasNext(); ) {
    Map.Entry entry = (Map.Entry) iter.next();
    Unit unit = (Unit) entry.getValue();

    Manifest manifest = new Manifest();
    Attributes attrs = new Attributes();
    manifest.getEntries().put(unit.getName(), attrs);

    attrs.put(new Attributes.Name("Manifest-Version"), "1.0");
    attrs.put(new Attributes.Name("JLego-Type"), "component");
    attrs.put(new Attributes.Name("JLego-Version"), this.VERSION);
    attrs.put(new Attributes.Name("Interface"), (unit.hasPackage() ?
unit.getPackage().getName() + '.' : "") + unit.getName());
    attrs.put(new Attributes.Name("Implementation"), (unit.hasPackage() ?
unit.getPackage().getName() + '.' : "") + unit.getName() + "Component");
    Attributes.Name COMP_DEP = new Attributes.Name("Component-Deps");
    attrs.put(COMP_DEP, "");
    Attributes.Name MOD_DEP = new Attributes.Name("Module-Deps");
    attrs.put(MOD_DEP, "");

    if (unit.hasVariables()) {
     Variables vars = unit.getVariables();
     for (int i = 0; i < vars.getSize(); i++) {
      Variable var = vars.getVariable(i);

      if (var.getMeta().equals("component")) {
       if (comp_files.containsKey(var.getType())) {
        /* copy from directory */
        List list = Files.getFileListing((File) comp_files.get(var.getType()), (File)
comp_files.get(var.getType()));
        Files.copy((File) comp_files.get(var.getType()), list, (File)
comp_files.get(unit.getName()));
       }
       else {
        /* copy from archive */
        try {
         Files.copy(new File(ClassPath.instance().findComponentPath(unit,
var.getType())), (File) comp_files.get(unit.getName()));
        }
        catch (ComponentNotFoundException e) {
         e.printStackTrace();
        }
       }

      /* add dependancy reference for this component */
```

```
          attrs.put(COMP_DEP, attrs.get(COMP_DEP) +
(attrs.get(COMP_DEP).equals("") ? "" : " ") + var.getType());
        }
        else if (var.getMeta().equals("module")) {
          /* add dependancy reference for this module */
          attrs.put(MOD_DEP, attrs.get(MOD_DEP) +
(attrs.get(MOD_DEP).equals("") ? "" : " ") + var.getType());
        }
       }
     }

     archive(unit, (File) comp_files.get(unit.getName()), manifest);
   }
  }

  /**
   * build component classpath referencing the precompiled modules upon which
it depends
   * @param classpath
   * @return
   */
  private String getComponentClasspath(String classpath) {
   StringBuffer comp_classpath = new StringBuffer(classpath);

   for (Iterator iter = components.entrySet().iterator(); iter.hasNext(); ) {
    Map.Entry entry = (Map.Entry) iter.next();
    Unit unit = (Unit) entry.getValue();

    /* get list of module variables */
    List mars = new ArrayList(5);
    if (unit.hasVariables()) {
     Variables vars = unit.getVariables();
     for (int i = 0; i < vars.getSize(); i++) {
      Variable var = vars.getVariable(i);

      if (var.getMeta().equals("module")) {
       try {

comp_classpath.append(File.pathSeparator).append(ClassPath.instance().findMod
ulePath(unit, var.getType()));
        }
        catch (ModuleNotFoundException e) {}
       }
       else if (var.getMeta().equals("component")) {
        try {
```

```
comp_classpath.append(File.pathSeparator).append(ClassPath.instance().findCom
ponentPath(unit, var.getType()));
          }
          catch (ComponentNotFoundException e) { }
        }
       }
     }
   }

   return comp_classpath.toString();
  }

  private File compile(String classpath, String name, String file) {
   File temp_dir = new File(System.getProperty("java.io.tmpdir") + File.separator
+ "jlego_" + name);
   temp_dir.mkdir();

   new File(temp_dir, "lib").mkdir();

   to_delete.add(temp_dir);

   List args = new ArrayList();
   args.add("-nowarn");
   args.add("-d");
   args.add(temp_dir.getPath());
   args.add("-sourcepath");
   args.add(source_dir.getPath());
   args.add("-classpath");
   args.add(classpath);
   args.add(source_dir.getPath() + File.separatorChar + file);

   sun.tools.javac.Main javac = new sun.tools.javac.Main(System.err, "jlego");
   javac.compile((String[]) args.toArray(new String[args.size()]));

   return temp_dir;
  }

  private void archive(Unit unit, File dir, Manifest manifest) {
   try {
    File archive = null;

    if (unit.hasPackage()) {
     archive = new File(build_dir, unit.getPackage().getPath());
     archive.mkdirs();
    }
```

```
    if (unit instanceof Component) {
      archive = new File(archive, unit.getName() +
Files.COMPONENT_ARCHIVE_EXT);
    }
    else {
      archive = new File(archive, unit.getName() +
Files.MODULE_ARCHIVE_EXT);
    }

    JarOutputStream out = new JarOutputStream(new FileOutputStream(archive),
manifest);
    out.setMethod(JarOutputStream.DEFLATED);
    out.setLevel(9);

    Files.archiveDirectory(out, dir, dir);

    out.finish();
    out.flush();
    out.close();
   }
   catch (IOException e) {
    e.printStackTrace();
   }
  }

  private void search(File dir) throws IOException {
   if (!dir.canRead() || !dir.canWrite()) {
    throw new JLegoError("cannot read/write " + dir.getAbsolutePath());
   }

   if (dir.isDirectory()) {
    File files[] = dir.listFiles();

    for (int i = 0; i < files.length; i++) {
     if (!files[i].canRead() || !files[i].canWrite()) {
      throw new JLegoError("cannot read/write " + files[i].getAbsolutePath());
     }

     if (files[i].isDirectory()) {
      search(files[i]);
     }
     else {
      if (Files.isCompilable(files[i])) {
       precompile(files[i]);
```

```
        String base = files[i].getPath().substring(0,
files[i].getPath().lastIndexOf('.'));
        if (Files.isComponent(files[i].getName())) {
          source_files.add(base + "Component" + Files.JAVA_EXT);
          source_files.add(base + Files.JAVA_EXT);
        }
        else if (Files.isModule(files[i].getName())) {
          source_files.add(base + "Module" + Files.JAVA_EXT);
          source_files.add(base + Files.JAVA_EXT);
        }
       }
       else if (Files.isJava(files[i].getName())) {
        source_files.add(files[i].getPath());
       }
      }
     }
    }
    else {
     precompile(dir);
    }
   }

   private ComponentCompiler getComponentCompiler() {
    if (comp_compiler == null) {
     this.comp_compiler = new ComponentCompiler();
    }

    return comp_compiler;
   }

   private ModuleCompiler getModuleCompiler() {
    if (mod_compiler == null) {
     this.mod_compiler = new ModuleCompiler();
    }

    return mod_compiler;
   }

   public static void printUsage(){
    System.out.println("JLego ~ Java Component Model");
    System.out.println("copyright 2003, Micah Spears");
    System.out.println();
    System.out.println("JLego was developed as a thesis project for Texas State
University");
    System.out.println();
    System.out.println("usage: jlego [options]");
```

```
    System.out.println();
    System.out.println("options:");
    System.out.println("\t-sourcepath\t\tthe relative or absolute path to your source
directory");
    System.out.println("\t-buildpath\t\tthe relative or absolute path to your build
directory");
    System.out.println("\t-classpath\t\tthe path used to resolve dependancies");
    System.out.println("\t-help\t\t\tprint the usage instructions");

    System.exit(0);
  }

  public static void main(String[] args) {
    String sourcepath = null;
    String classpath = "." + File.separator;
    String buildpath = null;

    for (int i = 0; i < args.length; i++) {
      if (args[i].equals("-help") && ((i + 1) < args.length)) {
        printUsage();
      }
      else if (args[i].equals("-sourcepath") && ((i + 1) < args.length)) {
        sourcepath = stripQuotes(args[++i]);
      }
      else if (args[i].equals("-classpath") && ((i + 1) < args.length)) {
        classpath += File.pathSeparatorChar + stripQuotes(args[++i]);
      }
      else if (args[i].equals("-buildpath") && ((i + 1) < args.length)) {
        buildpath = stripQuotes(args[++i]);
      }
    }

    if(sourcepath == null){
      System.out.println("-sourcepath must be specified!\n");
      printUsage();
    }

    if(buildpath == null){
      buildpath = sourcepath;
    }

    try {
      Compiler compiler = new Compiler(sourcepath, classpath, buildpath);
    }
    catch (CompileError e) {
      e.printStackTrace();
```

```
    }
  }

  private static String stripQuotes(String str){
    if(str.startsWith("\"") && str.endsWith("\"")){
      return str.substring(1, str.length() - 1);
    }

    return str;
  }

}


package org.swt.edu.jlego.compiler;

import java.io.*;
import java.util.*;

import org.swt.edu.jlego.model.*;
import org.swt.edu.jlego.parser.*;

public class ComponentCompiler extends AbstractCompiler {
  private Component comp;

  public ComponentCompiler() {
  }

  public Unit getUnit() {
    return comp;
  }

  public boolean execute() {
    FileInputStream in = null;

    try {
      in = new FileInputStream(file);

      Parser parser = Parser.createParser(in);
      this.comp = (Component) parser.Compilation();

      verifyName(comp);
      verifyPackage(comp);
      modify(comp);
      writeInterface(comp);
      writeUnit(comp);
```

```
      return true;
    }
    catch (Throwable e) {
      error = new CompileError("ComponentCompiler", "unable to compile \"" +
file.getPath() + "\"", e);
    }
    finally {
      try {
        if (in != null) {
          in.close();
        }
      }
      catch (Throwable t) { }
    }

    return false;
  }

  protected void modify(Unit unit) {
    super.modify(unit);

    if (!unit.hasConstructors()) {
      Constructor con = new Constructor();
      con.setOperation(new Operation());

      unit.addConstructor(con);
    }

    org.swt.edu.jlego.model.Exception condition = new
org.swt.edu.jlego.model.Exception("ConditionException");

    if (unit.hasMethods()) {
      Methods methods = unit.getMethods();
      for (int i = 0; i < methods.getSize(); i++) {
        Method method = methods.getMethod(i);
        method.addException(condition);
      }
    }

    if (unit.hasConstructors()) {
      Constructors cons = unit.getConstructors();
      for (int i = 0; i < cons.getSize(); i++) {
        Constructor con = cons.getConstructor(i);
        con.addException(condition);
      }
```

```
  }

  if (unit.hasDestructor()) {
   Destructor dest = unit.getDestructor();
   dest.addException(condition);
  }
 }

 protected void writeInterfaceHeader(PrintWriter out, Unit unit) {
  if (!unit.hasVariables()) {
   return;
  }

  Variables vars = unit.getVariables();

  for (int i = 0; i < vars.getSize(); i++) {
   Variable var = vars.getVariable(i);

   if (var.isModule()) {
    var.setAbsoluteType(resolveVariable(unit, var, true));

    out.print("  public interface ");
    out.print(var.getType());
    out.print(" extends ");
    out.print(var.getAbsoluteType());
    out.println(" {}");
    out.println();
   }
  }

  if (unit.hasDestructor()) {
   Destructor d = unit.getDestructor();
   out.print("  public void destroy()");

   if (d.hasExceptions()) {
    d.getExceptions().write(out);
   }

   out.println(";");
   out.println();
  }
 }

 protected void writeUnitFooter(PrintWriter out, Unit unit) {
  Component component = (Component) unit;
  if (component.hasInvariants()) {
```

```
Invariants invs = component.getInvariants();

out.println("  private void checkInvariants() {");

for (int i = 0; i < invs.getSize(); i++) {
  Invariant inv = invs.getInvariant(i);

  out.print("    ");
  out.print("if(!(");
  out.print(inv.getAssertion());
  out.println(")){");
  out.print("    ");
  out.print("  throw new InvariantException(\"");
  out.print(inv.getMessage());
  out.println("\");");
  out.print("    ");
  out.println("}");

  if ((i + 1) < invs.getSize()) {
    out.println();
  }
}

out.println("  }");
out.println();

Map map = new HashMap();

for (int i = 0; i < invs.getSize(); i++) {
  Invariant inv = invs.getInvariant(i);

  Variables vars = inv.getVariables();
  for (int j = 0; j < vars.getSize(); j++) {
    Variable var = vars.getVariable(j);

    if (!map.containsKey(var.getName())) {

      out.print("  private ");
      out.print(resolveVariable(getUnit(), var, true));
      out.print(" ");
      out.print(var.getName());
      out.println("() {");
      out.println("    checkInvariants();");
      out.println();
      out.print("    return ");
      out.print(var.getName());
```

```
        out.println(";");
        out.println("  }");
        out.println();

        map.put(var.getName(), var);
       }
      }
     }
    }
   }

  protected void writeMethodBody(PrintWriter out, Method method) {
   if (method.hasPostCondition()) {
    out.print("    ");
    out.print(method.getReturnType());
    out.print(" result = ");
    out.print(getInitializerForType(method.getReturnType()));
    out.println(";");

    out.println("    boolean has_ex_occured = false;");
    out.println();
   }

   out.println("    try {");

   if (method.hasPreCondition()) {
    writePreCondition(out, method.getPreCondition(), "      ");

    out.println();
   }

   Operation op = method.getOperation();

   if (op.hasStatements()) {
    Statements statements = op.getStatements();

    for (int i = 0; i < statements.getSize(); i++) {
     Statement statement = statements.get(i);

     statement.convert(this);

     if ((statement instanceof Statement.ReturnStatement) &&
method.hasPostCondition()) {
      out.print("      result = ");
      out.println(statement.getValue());
      out.println("      return result;");
```

```
        if ((i + 1) < statements.getSize()) {
          out.println();
        }
      }
      else {
        out.print("      ");
        statement.write(out);
      }
    }
  }

  out.println("    }");

  if (method.hasExceptions()) {
    writeCaughtExceptions(out, method.getExceptions(), "    ",
!method.getReturnType().equals("void"), method.hasPreCondition(),
method.hasPostCondition());
  }

  out.println("    catch(Exception e) {");

  if (method.hasPostCondition()) {
    out.println("      has_ex_occured = true;");
  }

  out.println("      throw new ConditionException(e.getMessage());");
  out.println("    }");

  if (method.hasPostCondition()) {
    out.println("    finally {");
    out.println("      try {");
    out.println("        if(!has_ex_occured) {");

    writePostCondition(out, method.getPostCondition(), "          ");

    out.println("        }");
    out.println("      }");
    out.println("      catch(ConditionException e) {");
    out.println("        throw e;");
    out.println("      }");
    out.println("      catch(Exception e) {");
    out.println("        throw new ConditionException(e.getMessage());");
    out.println("      }");
    out.println("    }");
  }
```

```
  }

  protected void writeConstructorBody(PrintWriter out, Constructor con) {
    out.println("    try {");

    if (con.hasPreCondition()) {
      writePreCondition(out, con.getPreCondition(), "      ");
      out.println();
    }

    writeOperation(out, con.getOperation(), "      ");

    if (con.hasPostCondition()) {
      out.println();
      writePostCondition(out, con.getPostCondition(), "      ");
    }

    out.println("    }");

    if (con.hasExceptions()) {
      writeCaughtExceptions(out, con.getExceptions(), "    ", false,
con.hasPreCondition(), con.hasPostCondition());
    }

    out.println("    catch(Exception e) {");
    out.println("      throw new ConditionException(e.getMessage());");
    out.println("    }");
  }

  protected void writeDestructorBody(PrintWriter out, Destructor d) {
    out.println("    try {");

    if (d.hasPreCondition()) {
      writePreCondition(out, d.getPreCondition(), "      ");
      out.println();
    }

    writeOperation(out, d.getOperation(), "      ");

    if (d.hasPostCondition()) {
      out.println();
      writePostCondition(out, d.getPostCondition(), "      ");
    }

    out.println("    }");
```

```
    if (d.hasExceptions()) {
      writeCaughtExceptions(out, d.getExceptions(), "    ", false,
d.hasPreCondition(), d.hasPostCondition());
    }

    out.println("    catch(Exception e) {");
    out.println("      throw new ConditionException(e.getMessage());");
    out.println("    }");
  }

  private void writeCaughtExceptions(PrintWriter out, Exceptions exps, String
space, boolean has_return, boolean has_pre_condition, boolean
has_post_condition) {
    for (Iterator iter = exps.iterator(); iter.hasNext(); ) {
      org.swt.edu.jlego.model.Exception e = (org.swt.edu.jlego.model.Exception)
iter.next();

      if (!e.isCatchable()) {
        continue;
      }

      if (!has_pre_condition && e.getName().equals("ConditionException")) {
        continue;
      }

      out.print(space);
      out.print("catch(");
      out.print(e.getName());
      out.println(" e) {");

      if (has_return && has_post_condition) {
        out.print(space);
        out.println("  has_ex_occured = true;");
      }

      out.print(space);
      out.println("  throw e;");
      out.print(space);
      out.println("}");
    }
  }

  private void writePostCondition(PrintWriter out, PostCondition post, String
space) {
    for (int i = 0; i < post.getSize(); i++) {
      Condition c = post.getCondition(i);
```

```
    out.print(space);
    out.print("if(!(");
    out.print(c.getAssertion());
    out.println(")){");
    out.print(space);
    out.print("  throw new PostConditionException(\"");
    out.print(c.getMessage());
    out.println("\");");
    out.print(space);
    out.println("}");
   }
  }

  private void writePreCondition(PrintWriter out, PreCondition pre, String space)
{
   for (int i = 0; i < pre.getSize(); i++) {
    Condition c = pre.getCondition(i);

    out.print(space);
    out.print("if(!(");
    out.print(c.getAssertion());
    out.println(")){");
    out.print(space);
    out.print("  throw new PreConditionException(\"");
    out.print(c.getMessage());
    out.println("\");");
    out.print(space);
    out.println("}");
   }
  }

}


package org.swt.edu.jlego.compiler;

public class ComponentNotFoundException extends Exception {

  public ComponentNotFoundException(String message) {
   super(message);
  }

}
```

```
package org.swt.edu.jlego.compiler;

import java.io.*;

import org.swt.edu.jlego.model.*;
import org.swt.edu.jlego.parser.*;

public class ModuleCompiler extends AbstractCompiler {
 private Module module;

 public ModuleCompiler() {
 }

 public Unit getUnit() {
  return module;
 }

 public boolean execute() {
  FileInputStream in = null;

  try {
   in = new FileInputStream(file);

   Parser parser = Parser.createParser(in);
   this.module = (Module) parser.Compilation();

   verifyName(module);
   verifyPackage(module);
   modify(module);
   writeInterface(module);
   writeUnit(module);

   return true;
  }
  catch (Throwable e) {
   error = new CompileError("ModuleCompiler", "unable to compile \"" +
file.getPath() + "\"", e);
  }
  finally {
   try {
    if (in != null) {
     in.close();
    }
   }
   catch (Throwable t) { }
  }
```

```
    return false;
  }

  protected void modify(Unit unit) {
    super.modify(unit);

    if (!unit.hasConstructors()) {
      Constructor con = new Constructor();
      con.setOperation(new Operation());

      unit.addConstructor(con);
    }
  }
}
```

```
package org.swt.edu.jlego.compiler;

public class ModuleNotFoundException extends Exception {

  public ModuleNotFoundException(String message) {
    super(message);
  }

}
```

Appendix H

```
package org.swt.edu.jlego.model;

import java.io.*;

public class Argument {
  private String value;

  public Argument() {
  }

  public Argument(String value) {
    this.value = value;
  }
```

```
  public void setValue(String value) {
    this.value = value.trim();
  }

  public String getValue() {
    return value;
  }

  public void write(PrintWriter writer) {
    writer.print(value);
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Arguments {
  private List args;

  public Arguments() {
    this.args = new ArrayList();
  }

  public void add(Argument param) {
    args.add(param);
  }

  public int getSize() {
    return args.size();
  }

  public Argument get(int index) {
    return (Argument) args.get(index);
  }

  public void write(PrintWriter writer) {
    for (int i = 0; i < getSize(); i++) {
      get(i).write(writer);

      if ((i + 1) < getSize()) {
        writer.print(", ");
      }
```

```
    }
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Component implements Unit {
  private Module module;
  private Invariants invariants;

  public Component() {
    this(null);
  }

  public Component(String name) {
    module = new Module(name);
  }

  public void setComposition(Composition comp) {
    module.setComposition(comp);
  }

  public Composition getComposition() {
    return module.getComposition();
  }

  public boolean hasComposition() {
    return module.hasComposition();
  }

  public void setPackage(Package pkg) {
    module.setPackage(pkg);
  }

  public Package getPackage() {
    return module.getPackage();
  }

  public boolean hasPackage() {
    return module.hasPackage();
  }
```

```
public void addImport(Import imp) {
 module.addImport(imp);
}

public void setImports(Imports imports) {
 module.setImports(imports);
}

public Imports getImports() {
 return module.getImports();
}

public boolean hasImports() {
 return module.hasImports();
}

public void setName(String name) {
 module.setName(name);
}

public String getName() {
 return module.getName();
}

public void addVariable(Variable var) {
 module.addVariable(var);
}

public void setVariables(Variables vars) {
 module.setVariables(vars);
}

public Variables getVariables() {
 return module.getVariables();
}

public boolean hasVariables() {
 return module.hasVariables();
}

public void setInvariants(Invariants invariants) {
 this.invariants = invariants;
}

public Invariants getInvariants() {
 return invariants;
```

```
}

public boolean hasInvariants() {
 return (invariants != null);
}

public void addConstructor(Constructor constructor) {
 module.addConstructor(constructor);
}

public void setConstructors(Constructors cons) {
 module.setConstructors(cons);
}

public Constructors getConstructors() {
 return module.getConstructors();
}

public boolean hasConstructors() {
 return module.hasConstructors();
}

public void setDestructor(Destructor destructor) {
 module.setDestructor(destructor);
}

public Destructor getDestructor() {
 return module.getDestructor();
}

public boolean hasDestructor() {
 return module.hasDestructor();
}

public void addMethod(Method method) {
 module.addMethod(method);
}

public void setMethods(Methods methods) {
 module.setMethods(methods);
}

public Methods getMethods() {
 return module.getMethods();
}
```

```
public boolean hasMethods() {
 return module.hasMethods();
}

public void write(PrintWriter writer) {
 if (hasPackage()) {
  getPackage().write(writer);
  writer.println();
 }

 if (hasImports()) {
  getImports().write(writer);
  writer.println();
 }

 writer.print("component ");
 writer.print(getName());

 if (hasComposition()) {
  getComposition().write(writer);
 }

 writer.println(" {");
 writer.println();

 if (hasVariables()) {
  getVariables().write(writer);
  writer.println();
 }

 if (hasInvariants()) {
  getInvariants().write(writer);
  writer.println();
 }

 if (hasConstructors()) {
  getConstructors().write(writer);
  writer.println();
 }

 if (hasDestructor()) {
  getDestructor().write(writer);
  writer.println();
 }

 if (hasMethods()) {
```

```
      getMethods().write(writer);
      writer.println();
    }

    writer.println("}");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Composite {
  private String name;

  public Composite() {
  }

  public Composite(String name) {
    this.name = name;
  }

  public void setName(String name) {
    this.name = name.trim();
  }

  public String getName() {
    return name;
  }

  public void write(PrintWriter writer) {
    writer.print(name);
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Composition {
  private List comps;
```

```
  public Composition() {
    this.comps = new ArrayList();
  }

  public void add(Composite comp) {
    comps.add(comp);
  }

  public int getSize() {
    return comps.size();
  }

  public Composite get(int index) {
    return (Composite) comps.get(index);
  }

  public void write(PrintWriter writer) {
    writer.print(" implements ");

    for (int i = 0; i < getSize(); i++) {
      get(i).write(writer);

      if ((i + 1) < getSize()) {
        writer.print(", ");
      }
    }
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Condition {
  private String message;
  private String assertion;

  public Condition() {
  }

  public Condition(String message, String assertion) {
    this.message = message;
    this.assertion = assertion;
```

```
  }

  public String getMessage() {
   return message;
  }

  public void setMessage(String message) {
   if (message.startsWith("\"") && message.endsWith("\"")) {
    message = message.substring(1, message.length() - 1);
   }

   this.message = message;
  }

  public String getAssertion() {
   return assertion;
  }

  public void setAssertion(String assertion) {
   this.assertion = assertion;
  }

  public void write(PrintWriter writer) {
   writer.print("condition(");
   writer.print(assertion);
   writer.print(", \"");
   writer.print(message);
   writer.println("\");");
  }
}


package org.swt.edu.jlego.model;

import java.io.*;

public class Constructor {
 private Method method;

 public Constructor() {
  this.method = new Method("", null);
 }

 public void addParameter(Parameter param) {
  method.addParameter(param);
 }
```

```
public Parameters getParameters() {
 return method.getParameters();
}

public boolean hasParameters() {
 return method.hasParameters();
}

public void addException(Exception exp) {
 method.addException(exp);
}

public Exceptions getExceptions() {
 return method.getExceptions();
}

public boolean hasExceptions() {
 return method.hasExceptions();
}

public void setOperation(Operation operation) {
 method.setOperation(operation);
}

public Operation getOperation() {
 return method.getOperation();
}

public boolean hasOperation() {
 return method.hasOperation();
}

public void setPreCondition(PreCondition pre) {
 method.setPreCondition(pre);
}

public PreCondition getPreCondition() {
 return method.getPreCondition();
}

public boolean hasPreCondition() {
 return method.hasPreCondition();
}

public void setPostCondition(PostCondition post) {
```

```
 method.setPostCondition(post);
}

public PostCondition getPostCondition() {
 return method.getPostCondition();
}

public boolean hasPostCondition() {
 return method.hasPostCondition();
}

public void write(PrintWriter writer) {
 writer.print("  constructor(");

 if (hasParameters()) {
  method.getParameters().write(writer);
 }

 writer.print(")");

 if (hasExceptions()) {
  method.getExceptions().write(writer);
  writer.print(" ");
 }

 writer.println("{");
 writer.println();

 if (hasPreCondition()) {
  method.getPreCondition().write(writer);
  writer.println();
 }

 if (hasPostCondition()) {
  method.getPostCondition().write(writer);
  writer.println();
 }

 if (hasOperation()) {
  method.getOperation().write(writer);
  writer.println();
 }

 writer.println("  }");
}
```

```
}


package org.swt.edu.jlego.model;

import java.io.*;

public class ConstructorInvocation {
 private Arguments args;

 public ConstructorInvocation() {
  args = new Arguments();
 }

 public void addArgument(Argument argue) {
  this.args.add(argue);
 }

 public void setArguments(Arguments args) {
  this.args = args;
 }

 public Arguments getArguments() {
  return args;
 }

 public boolean hasArguements() {
  return ((args != null) && (args.getSize() > 0));
 }

 public void write(PrintWriter writer) {
  writer.print("this(");

  if (hasArguements()) {
   args.write(writer);
  }

  writer.println(");");
 }
}



package org.swt.edu.jlego.model;

import java.io.*;
```

```
import java.util.*;

public class Constructors {
 private List constructors;

 public Constructors() {
  constructors = new ArrayList();
 }

 public void add(Constructor constructor) {
  constructors.add(constructor);
 }

 public int getSize() {
  return constructors.size();
 }

 public Constructor getConstructor(int index) {
  return (Constructor) constructors.get(index);
 }

 public void addException(Exception e) {
  for (int i = 0; i < getSize(); i++) {
   getConstructor(i).addException(e);
  }
 }

 public void write(PrintWriter writer) {
  for (int i = 0; i < getSize(); i++) {
   getConstructor(i).write(writer);

   if ((i + 1) < getSize()) {
    writer.println();
   }
  }
 }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Destructor {
 private Method method;
```

```
public Destructor() {
  this.method = new Method("", null);
}

public void setOperation(Operation operation) {
  method.setOperation(operation);
}

public Operation getOperation() {
  return method.getOperation();
}

public boolean hasOperation() {
  return method.hasOperation();
}

public void setPreCondition(PreCondition pre) {
  method.setPreCondition(pre);
}

public PreCondition getPreCondition() {
  return method.getPreCondition();
}

public boolean hasPreCondition() {
  return method.hasPreCondition();
}

public void setPostCondition(PostCondition post) {
  method.setPostCondition(post);
}

public PostCondition getPostCondition() {
  return method.getPostCondition();
}

public boolean hasPostCondition() {
  return method.hasPostCondition();
}

public void addException(Exception exp) {
  method.addException(exp);
}

public Exceptions getExceptions() {
```

```
    return method.getExceptions();
  }

  public boolean hasExceptions() {
    return method.hasExceptions();
  }

  public void write(PrintWriter writer) {
    writer.println("  destructor {");
    writer.println();

    if (hasPreCondition()) {
      method.getPreCondition().write(writer);
      writer.println();
    }

    if (hasPostCondition()) {
      method.getPostCondition().write(writer);
      writer.println();
    }

    if (hasOperation()) {
      method.getOperation().write(writer);
      writer.println();
    }

    writer.println("  }");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Exception {
  private String name;

  public Exception() {
  }

  public Exception(String name) {
    this.name = name;
  }
```

```
  public void setName(String name) {
   this.name = name;
  }

  public String getName() {
   return name;
  }

  public boolean isCatchable() {
   if ("java.lang.Exception".equals(name) || "Exception".equals(name)) {
    return false;
   }

   if ("java.lang.Throwable".equals(name) || "Throwable".equals(name)) {
    return false;
   }

   return true;
  }

  public void write(PrintWriter writer) {
   writer.print(name);
  }

  public int hashCode() {
   return getName().hashCode();
  }

  public boolean equals(Object obj) {
   if ((obj == null) || !(obj instanceof Exception)) {
    return false;
   }

   return ((Exception) obj).getName().equals(getName());
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Exceptions {
  private Set exps;
```

```
public Exceptions() {
 this.exps = new HashSet();
}

public void add(Exception exp) {
 exps.add(exp);
}

public int getSize() {
 return exps.size();
}

public Iterator iterator() {
 return exps.iterator();
}

public Exceptions getUserExceptions() {
 Exceptions exps = new Exceptions();

 for (Iterator iter = iterator(); iter.hasNext(); ) {
  Exception e = (Exception) iter.next();

  if (e.getName().equals("ConditionException")) {
   continue;
  }

  exps.add(e);
 }

 return exps;
}

public void write(PrintWriter writer) {
 writer.print(" throws ");

 for (Iterator iter = iterator(); iter.hasNext(); ) {
  Exception e = (Exception) iter.next();
  e.write(writer);

  if (iter.hasNext()) {
   writer.print(", ");
  }
 }
}
```

```
}


package org.swt.edu.jlego.model;

import java.io.*;

public class Import {
 private String name;

 public Import() {
 }

 public Import(String name) {
  this.name = name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public String getName() {
  return name;
 }

 public boolean isGlobal() {
  return name.endsWith(".*");
 }

 public String getPath() {
  if (isGlobal()) {
    return name.substring(0, name.length() - 2);
  }

  return name;
 }

 public void write(PrintWriter writer) {
  writer.print("import ");
  writer.print(name);
  writer.println(";");
 }

 public int hashCode() {
  return getName().hashCode();
 }
```

```
  public boolean equals(Object obj) {
    if ((obj == null) || !(obj instanceof Import)) {
      return false;
    }

    return ((Import) obj).getName().equals(getName());
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Imports {
  private Set imps;

  public Imports() {
    this.imps = new HashSet();
  }

  public void add(Import imp) {
    imps.add(imp);
  }

  public int getSize() {
    return imps.size();
  }

  public Iterator iterator() {
    return imps.iterator();
  }

  public void write(PrintWriter writer) {
    for (Iterator iter = iterator(); iter.hasNext(); ) {
      ((Import) iter.next()).write(writer);
    }
  }

}


package org.swt.edu.jlego.model;
```

```
public class Invariant extends Condition {
 private Variables vars;

 public Invariant() {
  vars = new Variables();
 }

 public void addVariable(Variable var) {
  vars.add(var);
 }

 public Variables getVariables() {
  return vars;
 }

 public boolean hasVariables() {
  return (vars.getSize() > 0);
 }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Invariants {
 private List invs;

 public Invariants() {
  invs = new ArrayList();
 }

 public void add(Invariant inv) {
  invs.add(inv);
 }

 public int getSize() {
  return invs.size();
 }

 public Invariant getInvariant(int index) {
  return (Invariant) invs.get(index);
 }
```

```
  public void write(PrintWriter writer) {
    writer.println("  invariants {");

    for (int i = 0; i < getSize(); i++) {
      writer.print("    ");
      getInvariant(i).write(writer);
    }

    writer.println("  }");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Method {
  private String name;
  private String return_type;
  private Parameters params;
  private Exceptions exps;
  private Operation operation;
  private PreCondition pre;
  private PostCondition post;

  public Method() {
    this(null, null);
  }

  public Method(String return_type, String name) {
    this.return_type = return_type;
    this.name = name;
    this.params = new Parameters();
    this.exps = new Exceptions();
  }

  public void setReturnType(String type) {
    this.return_type = type;
  }

  public String getReturnType() {
    return return_type;
  }
```

```
public void setName(String name) {
 this.name = name;
}

public String getName() {
 return name;
}

public void addParameter(Parameter param) {
 this.params.add(param);
}

public Parameters getParameters() {
 return params;
}

public boolean hasParameters() {
 return ((params != null) && (params.getSize() > 0));
}

public void addException(Exception exp) {
 this.exps.add(exp);
}

public Exceptions getExceptions() {
 return exps;
}

public boolean hasExceptions() {
 return ((exps != null) && (exps.getSize() > 0));
}

public void setOperation(Operation operation) {
 this.operation = operation;
}

public Operation getOperation() {
 return operation;
}

public boolean hasOperation() {
 return (operation != null);
}

public void setPreCondition(PreCondition pre) {
```

```
 this.pre = pre;
}

public PreCondition getPreCondition() {
 return pre;
}

public boolean hasPreCondition() {
 return (pre != null);
}

public void setPostCondition(PostCondition post) {
 this.post = post;
}

public PostCondition getPostCondition() {
 return post;
}

public boolean hasPostCondition() {
 return (post != null);
}

public void write(PrintWriter writer) {
 writer.print("  method ");
 writer.print(return_type);
 writer.print(" ");
 writer.print(name);
 writer.print("(");

 if (hasParameters()) {
  params.write(writer);
 }

 writer.print(") ");

 if (hasExceptions()) {
  exps.write(writer);
  writer.print(" ");
 }

 writer.println("{");
 writer.println();

 if (hasPreCondition()) {
  pre.write(writer);
```

```
      writer.println();
    }

    if (hasPostCondition()) {
      post.write(writer);
      writer.println();
    }

    if (hasOperation()) {
      operation.write(writer);
      writer.println();
    }

    writer.println("  }");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Methods {
  private List methods;

  public Methods() {
    methods = new ArrayList();
  }

  public void add(Method method) {
    methods.add(method);
  }

  public int getSize() {
    return methods.size();
  }

  public Method getMethod(int index) {
    return (Method) methods.get(index);
  }

  public void addException(Exception e) {
    for (int i = 0; i < getSize(); i++) {
      getMethod(i).addException(e);
```

```
    }
  }

  public void write(PrintWriter writer) {
    for (int i = 0; i < getSize(); i++) {
      getMethod(i).write(writer);

      if ((i + 1) < getSize()) {
        writer.println();
      }
    }
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Module implements Unit {
  private Package pkg;
  private Imports imports;
  private String name;
  private Composition composition;
  private Variables variables;
  private Constructors constructors;
  private Destructor destructor;
  private Methods methods;

  public Module() {
    this(null);
  }

  public Module(String name) {
    this.name = name;

    this.imports = new Imports();
    this.variables = new Variables();
    this.constructors = new Constructors();
    this.methods = new Methods();
  }

  public void setPackage(Package pkg) {
    this.pkg = pkg;
  }
```

```
public Package getPackage() {
 return pkg;
}

public boolean hasPackage() {
 return (pkg != null);
}

public void addImport(Import imp) {
 imports.add(imp);
}

public void setImports(Imports imports) {
 this.imports = imports;
}

public Imports getImports() {
 return imports;
}

public boolean hasImports() {
 return (imports.getSize() > 0);
}

public void setName(String name) {
 this.name = name;
}

public String getName() {
 return name;
}

public void setComposition(Composition comp) {
 this.composition = comp;
}

public Composition getComposition() {
 return composition;
}

public boolean hasComposition() {
 return (composition != null);
}

public void addVariable(Variable var) {
```

```
  variables.add(var);
}

public void setVariables(Variables vars) {
  this.variables = vars;
}

public Variables getVariables() {
  return variables;
}

public boolean hasVariables() {
  return (variables.getSize() > 0);
}

public void addConstructor(Constructor constructor) {
  constructors.add(constructor);
}

public void setConstructors(Constructors cons) {
  this.constructors = cons;
}

public Constructors getConstructors() {
  return constructors;
}

public boolean hasConstructors() {
  return (constructors.getSize() > 0);
}

public void setDestructor(Destructor destructor) {
  this.destructor = destructor;
}

public Destructor getDestructor() {
  return destructor;
}

public boolean hasDestructor() {
  return (destructor != null);
}

public void addMethod(Method method) {
  methods.add(method);
}
```

```
public void setMethods(Methods methods) {
 this.methods = methods;
}

public Methods getMethods() {
 return methods;
}

public boolean hasMethods() {
 return (methods.getSize() > 0);
}

public void write(PrintWriter writer) {
 if (hasPackage()) {
  getPackage().write(writer);
  writer.println();
 }

 if (hasImports()) {
  getImports().write(writer);
  writer.println();
 }

 writer.print("module ");
 writer.print(getName());

 if (hasComposition()) {
  getComposition().write(writer);
 }

 writer.println(" {");
 writer.println();

 if (hasVariables()) {
  getVariables().write(writer);
  writer.println();
 }

 if (hasConstructors()) {
  getConstructors().write(writer);
  writer.println();
 }

 if (hasDestructor()) {
  getDestructor().write(writer);
```

```
    writer.println();
   }

   if (hasMethods()) {
    getMethods().write(writer);
    writer.println();
   }

   writer.println("}");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Operation {
 private Statements statements;
 private ConstructorInvocation invocation;

 public Operation() {
  this.statements = new Statements();
 }

 public void addStatement(Statement s) {
  statements.add(s);
 }

 public Statements getStatements() {
  return statements;
 }

 public boolean hasStatements() {
  return (statements.getSize() > 0);
 }

 public void setConstructorInvocation(ConstructorInvocation invo) {
  this.invocation = invo;
 }

 public ConstructorInvocation getConstructorInvocation() {
  return invocation;
 }
```

```
  public boolean hasConstructorInvocation() {
    return (invocation != null);
  }

  public void write(PrintWriter writer) {
    writer.print("    operation {\n");

    if (hasConstructorInvocation()) {
      writer.print("      ");
      invocation.write(writer);
      writer.println();
    }

    if (hasStatements()) {
      statements.write(writer, "      ");
    }

    writer.println("    }");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Package {
  private String name;

  public Package() {
  }

  public Package(String name) {
    this.name = name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public String getPath() {
```

```
    return name.replace('.', '/');
  }

  public void write(PrintWriter writer) {
    writer.print("package ");
    writer.print(name);
    writer.println(";");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;

public class Parameter {
  private String type;
  private String name;

  public Parameter() {
  }

  public Parameter(String type, String name) {
    this.type = type;
    this.name = name;
  }

  public void setType(String type) {
    this.type = type;
  }

  public String getType() {
    return type;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void write(PrintWriter writer) {
    writer.print(type);
```

```
    writer.print(" ");
    writer.print(name);
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Parameters {
  private List params;

  public Parameters() {
    this.params = new ArrayList();
  }

  public void add(Parameter param) {
    params.add(param);
  }

  public int getSize() {
    return params.size();
  }

  public Parameter get(int index) {
    return (Parameter) params.get(index);
  }

  public void writeVariables(PrintWriter writer) {
    for (int i = 0; i < getSize(); i++) {
      writer.print(get(i).getName());

      if ((i + 1) < getSize()) {
        writer.print(", ");
      }
    }
  }

  public void write(PrintWriter writer) {
    for (int i = 0; i < getSize(); i++) {
      get(i).write(writer);

      if ((i + 1) < getSize()) {
```

```
      writer.print(", ");
     }
    }
   }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class PostCondition {
 private List conditions;

 public PostCondition() {
  conditions = new ArrayList();
 }

 public void add(Condition condition) {
  conditions.add(condition);
 }

 public int getSize() {
  return conditions.size();
 }

 public Condition getCondition(int index) {
  return (Condition) conditions.get(index);
 }

 public void write(PrintWriter writer) {
  writer.println("    post {");

  for (int i = 0; i < getSize(); i++) {
   writer.print("      ");
   getCondition(i).write(writer);
  }

  writer.println("    }");
 }

}
```

```
package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class PreCondition {
 private List conditions;

 public PreCondition() {
  conditions = new ArrayList();
 }

 public void add(Condition condition) {
  conditions.add(condition);
 }

 public int getSize() {
  return conditions.size();
 }

 public Condition getCondition(int index) {
  return (Condition) conditions.get(index);
 }

 public void write(PrintWriter writer) {
  writer.println("    pre {");

  for (int i = 0; i < getSize(); i++) {
   writer.print("      ");
   getCondition(i).write(writer);
  }

  writer.println("    }");
 }

}


package org.swt.edu.jlego.model;

import java.io.*;

import org.swt.edu.jlego.compiler.*;

public abstract class Statement {
 private String value;
```

```
 public Statement() {
  this(null);
 }

 public void convert(AbstractCompiler compiler) {
  if ((compiler instanceof ComponentCompiler) && ((Component)
compiler.getUnit()).hasInvariants()) {
    Variables vars = compiler.getUnit().getVariables();

    for (int i = 0; i < vars.getSize(); i++) {
     Variable var = vars.getVariable(i);
     setValue(getValue().replaceAll(var.getName(), var.getName() + "()"));
    }
  }
 }

 public Statement(String value) {
  setValue(value);
 }

 public String getValue() {
  return value;
 }

 public void setValue(String value) {
  this.value = value.trim();
 }

 public void write(PrintWriter writer) {
  writer.println(value);
 }

 public static class NormalStatement extends Statement {

  public NormalStatement() {
  }

  public NormalStatement(String value) {
   super(value);
  }
 }


 public static class LoadStatement extends Statement {
```

```
  public LoadStatement() {
  }

  public LoadStatement(String value) {
   super(value);
  }

  public void convert(AbstractCompiler compiler) {
   StringBuffer buffer = new StringBuffer(getValue());

   try {
    ClassPath.instance().findModule(compiler.getUnit(), getName(getValue()));
    buffer.insert(buffer.lastIndexOf("("), "Module.create");
   }
   catch (ModuleNotFoundException e) {

    if (compiler instanceof ComponentCompiler) {
     try {
      ClassPath.instance().findComponent(compiler.getUnit(),
getName(getValue()));
      buffer.insert(buffer.lastIndexOf("("), "Component.create");
     }
     catch (ComponentNotFoundException ex) {}
    }
   }

   int index = buffer.lastIndexOf("load");
   buffer.replace(index, index + "load".length() + 1, "");

   setValue(buffer.toString());
  }

  private String getName(String statement) {
   int end = statement.lastIndexOf('(');
   int start = end;
   while (statement.charAt(start - 1) != ' ' && start > 0) {
    start--;
   }

   return statement.substring(start, end);
  }
 }


 public static class NewStatement extends Statement {
```

```
    public NewStatement() {
    }

    public NewStatement(String value) {
      super(value);
    }
  }


  public static class ReturnStatement extends Statement {

    public ReturnStatement() {
    }

    public ReturnStatement(String value) {
      super(value);
    }

    public void write(PrintWriter writer) {
      writer.print("return ");

      super.write(writer);
    }
  }


  public static class AssignmentStatement extends Statement {

    public AssignmentStatement() {
    }

    public AssignmentStatement(String value) {
      super(value);
    }

    public void convert(AbstractCompiler compiler) { }
  }

}



package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;
```

```
public class Statements {
 private List statements;

 public Statements() {
  this.statements = new ArrayList();
 }

 public void add(Statement s) {
  statements.add(s);
 }

 public int getSize() {
  return statements.size();
 }

 public Statement get(int index) {
  return (Statement) statements.get(index);
 }

 public void write(PrintWriter writer, String space) {
  for (int i = 0; i < getSize(); i++) {
   writer.print(space);
   get(i).write(writer);
  }
 }

}


package org.swt.edu.jlego.model;

import java.io.*;

public interface Unit {

 public void setPackage(Package pkg);

 public Package getPackage();

 public boolean hasPackage();

 public void addImport(Import imp);

 public void setImports(Imports imports);
```

```
public Imports getImports();

public boolean hasImports();

public void setName(String name);

public String getName();

public void addVariable(Variable var);

public void setVariables(Variables vars);

public Variables getVariables();

public boolean hasVariables();

public void addConstructor(Constructor constructor);

public void setConstructors(Constructors cons);

public Constructors getConstructors();

public boolean hasConstructors();

public void setDestructor(Destructor destructor);

public Destructor getDestructor();

public boolean hasDestructor();

public void addMethod(Method method);

public void setMethods(Methods methods);

public Methods getMethods();

public boolean hasMethods();

public void setComposition(Composition comp);

public Composition getComposition();

public boolean hasComposition();

public void write(PrintWriter writer);
}
```

```
package org.swt.edu.jlego.model;

import java.io.*;

public class Variable {
 private String type;
 private String abs_type;
 private String name;
 private String meta;
 private String assign_type;
 private String assign_name;
 private Parameters assign_params;

 public Variable() {
  this(null, null, null);
 }

 public Variable(String meta, String type, String name) {
  this.meta = meta;
  this.type = type;
  this.name = name;

  this.assign_params = new Parameters();
 }

 public void setType(String type) {
  this.type = type;
 }

 public String getType() {
  return type;
 }

 public void setAbsoluteType(String type) {
  this.abs_type = type;
 }

 public String getAbsoluteType() {
  return abs_type;
 }

 public void setName(String name) {
  this.name = name;
 }
```

```
public String getName() {
 return name;
}

public void setMeta(String meta) {
 this.meta = meta;
}

public String getMeta() {
 return meta;
}

public boolean isModule() {
 return "module".equals(meta);
}

public boolean isComponent() {
 return "component".equals(meta);
}

public boolean isClass() {
 return "class".equals(meta);
}

public boolean hasAssignment() {
 return ((assign_type != null));
}

public void setAssignmentType(String type) {
 this.assign_type = type;
}

public String getAssignmentType() {
 return assign_type;
}

public void setAssignmentName(String name) {
 this.assign_name = name;
}

public String getAssignmentName() {
 return assign_name;
}

public void addAssignmentParameter(Parameter param) {
```

```
    this.assign_params.add(param);
  }

  public Parameters getAssignmentParameters() {
    return assign_params;
  }

  public boolean hasAssignmentParameters() {
    return ((assign_params != null) && (assign_params.getSize() > 0));
  }

  public void write(PrintWriter writer) {
    writer.print("  ");
    writer.print(meta);
    writer.print(" ");
    writer.print(type);
    writer.print(" ");
    writer.print(name);

    if (hasAssignment()) {
      writer.print(" = ");
      writer.print(assign_type);
      writer.print(" ");
      writer.print(assign_name);
      writer.print("(");

      if (hasAssignmentParameters()) {
        assign_params.write(writer);
      }

      writer.print(")");
    }

    writer.println(";");
  }

}


package org.swt.edu.jlego.model;

import java.io.*;
import java.util.*;

public class Variables {
  private List vars;
```

```
  public Variables() {
    this.vars = new ArrayList();
  }

  public void add(Variable var) {
    vars.add(var);
  }

  public int getSize() {
    return vars.size();
  }

  public Variable getVariable(int index) {
    return (Variable) vars.get(index);
  }

  public void write(PrintWriter writer) {
    for (int i = 0; i < getSize(); i++) {
      getVariable(i).write(writer);
    }
  }

}
```

Appendix I

```
options {
  JAVA_UNICODE_ESCAPE = true;
  STATIC = false;
  ERROR_REPORTING = true;
}

PARSER_BEGIN(Parser)
  package org.swt.edu.jlego.parser;

  import java.util.*;
  import java.io.*;

  import org.swt.edu.jlego.model.*;
  import org.swt.edu.jlego.*;

  public class Parser {
    private static Parser parser;
```

```
  public static Parser createParser(InputStream in){
    if(parser == null){
      parser = new Parser(in);
    }
    else{
      parser.ReInit(in);
    }

    return parser;
  }

  private boolean is_module;
  private boolean is_load_statement;
  private boolean is_new_statement;
  private boolean is_assignment_statement;

  private StringBuffer traverse(Token head, Token tail){
    StringBuffer buff = new StringBuffer(32);

    for(Token curr = head; curr != tail ; curr = curr.next) {
      includeSpecial(buff, curr);

      buff.append(curr.image);
    }

    return buff;
  }

  private void includeSpecial(StringBuffer buff, Token t){
    Token special = t.specialToken;

    if (special != null){
      while (special.specialToken != null){
        special = special.specialToken;
      }

      while(special != null){
        /* strip comments */
        if(special.kind != ParserConstants.SINGLE_LINE_COMMENT &&
special.kind != ParserConstants.MULTI_LINE_COMMENT){
          buff.append(special.image);
        }

        special = special.next;
      }
```

```
    }
   }

   private Variables parseVariables(Token head, Token tail){
    Variables vars = new Variables();

    for(Token curr = head; curr != tail ; curr = curr.next) {
     if(curr.kind == ParserConstants.IDENTIFIER){
      if((curr.next != tail) && (curr.next.kind != ParserConstants.LPAREN)){
       vars.add(new Variable(null, null, curr.image));
      }
     }
    }

    return vars;
   }

 }

PARSER_END(Parser)


/* whitespace SKIP : */
SPECIAL_TOKEN :
{
 " "
| "\t"
| "\n"
| "\r"
| "\f"
}

/* comments */

MORE :
{
 "//" : IN_SINGLE_LINE_COMMENT
|
 <"/**" ~["/"]> { input_stream.backup(1); } : IN_FORMAL_COMMENT
|
 "/*" : IN_MULTI_LINE_COMMENT
}

<IN_SINGLE_LINE_COMMENT>
SPECIAL_TOKEN :
{
```

```
 <SINGLE_LINE_COMMENT: "\n" | "\r" | "\r\n" > : DEFAULT
}

<IN_FORMAL_COMMENT>
SPECIAL_TOKEN :
{
 <FORMAL_COMMENT: "*/" > : DEFAULT
}

<IN_MULTI_LINE_COMMENT>
SPECIAL_TOKEN :
{
 <MULTI_LINE_COMMENT: "*/" > : DEFAULT
}

<IN_SINGLE_LINE_COMMENT,IN_FORMAL_COMMENT,IN_MULTI_LIN
E_COMMENT>
MORE :
{
 < ~[] >
}

/* component reserved words */

TOKEN :
{
 < MODULE: "module" >
| < COMPONENT: "component" >
| < PRE: "pre" >
| < POST: "post" >
| < OPERATION: "operation" >
| < INVARIANTS: "invariants" >
| < INVARIANT: "invariant" >
| < CONDITION: "condition" >
| < CONSTRUCTOR: "constructor" >
| < DECONSTRUCTOR: "destructor" >
| < METHOD: "method" >
| < RESULT: "result" >
| < LOAD: "load" >
}

/* java reserved words */

TOKEN :
{
 < BOOLEAN: "boolean" >
```

```
| < BREAK: "break" >
| < BYTE: "byte" >
| < CASE: "case" >
| < CATCH: "catch" >
| < CHAR: "char" >
| < CONST: "const" >
| < CONTINUE: "continue" >
| < _DEFAULT: "default" >
| < DO: "do" >
| < DOUBLE: "double" >
| < ELSE: "else" >
| < FALSE: "false" >
| < FINALLY: "finally" >
| < FLOAT: "float" >
| < FOR: "for" >
| < IF: "if" >
| < IMPLEMENTS: "implements" >
| < IMPORT: "import" >
| < INSTANCEOF: "instanceof" >
| < INT: "int" >
| < LONG: "long" >
| < NULL: "null" >
| < PACKAGE: "package">
| < RETURN: "return" >
| < SHORT: "short" >
| < SWITCH: "switch" >
| < SYNCHRONIZED: "synchronized" >
| < THIS: "this" >
| < THROW: "throw" >
| < THROWS: "throws" >
| < TRUE: "true" >
| < TRY: "try" >
| < VOID: "void" >
| < WHILE: "while" >
| < NEW: "new" >
| < FINAL: "final" >
| < CLASS: "class" >
}

/* literals */

TOKEN :
{
 < INTEGER_LITERAL:
      <DECIMAL_LITERAL> (["l","L"])?
    | <HEX_LITERAL> (["l","L"])?
```

```
    | <OCTAL_LITERAL> (["l","L"])?
  >
|
  < #DECIMAL_LITERAL: ["1"-"9"] (["0"-"9"])* >
|
  < #HEX_LITERAL: "0" ["x","X"] (["0"-"9","a"-"f","A"-"F"])+ >
|
  < #OCTAL_LITERAL: "0" (["0"-"7"])* >
|
  < FLOATING_POINT_LITERAL:
      (["0"-"9"])+ "." (["0"-"9"])* (<EXPONENT>)? (["f","F","d","D"])?
    | "." (["0"-"9"])+ (<EXPONENT>)? (["f","F","d","D"])?
    | (["0"-"9"])+ <EXPONENT> (["f","F","d","D"])?
    | (["0"-"9"])+ (<EXPONENT>)? ["f","F","d","D"]
  >
|
  < #EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+ >
|
  < CHARACTER_LITERAL:
    "'"
    (   (~["'","\\","\n","\r"])
      | ("\\"
          ( ["n","t","b","r","f","\\","'","\""]
          | ["0"-"7"] ( ["0"-"7"] )?
          | ["0"-"3"] ["0"-"7"] ["0"-"7"]
          )
        )
    )
    "'"
  >
|
  < STRING_LITERAL:
    "\""
    (   (~["\"","\\","\n","\r"])
      | ("\\"
          ( ["n","t","b","r","f","\\","'","\""]
          | ["0"-"7"] ( ["0"-"7"] )?
          | ["0"-"3"] ["0"-"7"] ["0"-"7"]
          )
        )
    )*
    "\""
  >
}

/* IDENTIFIERS */
```

```
TOKEN :
{
 < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
|
 < #LETTER:
    [
     "\u0024",
     "\u0041"-"\u005a",
     "\u005f",
     "\u0061"-"\u007a",
     "\u00c0"-"\u00d6",
     "\u00d8"-"\u00f6",
     "\u00f8"-"\u00ff",
     "\u0100"-"\u1fff",
     "\u3040"-"\u318f",
     "\u3300"-"\u337f",
     "\u3400"-"\u3d2d",
     "\u4e00"-"\u9fff",
     "\uf900"-"\ufaff"
    ]
 >
|
 < #DIGIT:
    [
     "\u0030"-"\u0039",
     "\u0660"-"\u0669",
     "\u06f0"-"\u06f9",
     "\u0966"-"\u096f",
     "\u09e6"-"\u09ef",
     "\u0a66"-"\u0a6f",
     "\u0ae6"-"\u0aef",
     "\u0b66"-"\u0b6f",
     "\u0be7"-"\u0bef",
     "\u0c66"-"\u0c6f",
     "\u0ce6"-"\u0cef",
     "\u0d66"-"\u0d6f",
     "\u0e50"-"\u0e59",
     "\u0ed0"-"\u0ed9",
     "\u1040"-"\u1049"
    ]
 >
}

/* SEPARATORS */
```

```
TOKEN :
{
  < LPAREN: "(" >
| < RPAREN: ")" >
| < LBRACE: "{" >
| < RBRACE: "}" >
| < LBRACKET: "[" >
| < RBRACKET: "]" >
| < SEMICOLON: ";" >
| < COMMA: "," >
| < DOT: "." >
}

/* OPERATORS */

TOKEN :
{
  < ASSIGN: "=" >
| < GT: ">" >
| < LT: "<" >
| < BANG: "!" >
| < TILDE: "~" >
| < HOOK: "?" >
| < COLON: ":" >
| < EQ: "==" >
| < LE: "<=" >
| < GE: ">=" >
| < NE: "!=" >
| < SC_OR: "||" >
| < SC_AND: "&&" >
| < INCR: "++" >
| < DECR: "--" >
| < PLUS: "+" >
| < MINUS: "-" >
| < STAR: "*" >
| < SLASH: "/" >
| < BIT_AND: "&" >
| < BIT_OR: "|" >
| < XOR: "^" >
| < REM: "%" >
| < LSHIFT: "<<" >
| < RSIGNEDSHIFT: ">>" >
| < RUNSIGNEDSHIFT: ">>>" >
| < PLUSASSIGN: "+=" >
| < MINUSASSIGN: "-=" >
| < STARASSIGN: "*=" >
```

```
| < SLASHASSIGN: "/=" >
| < ANDASSIGN: "&=" >
| < ORASSIGN: "|=" >
| < XORASSIGN: "^=" >
| < REMASSIGN: "%=" >
| < LSHIFTASSIGN: "<<=" >
| < RSIGNEDSHIFTASSIGN: ">>=" >
| < RUNSIGNEDSHIFTASSIGN: ">>>=" >
}

/*****************************************
 * The Component Language Grammar starts here *
 *****************************************/

/* program structuring syntax */

Unit Compilation() :
{
  org.swt.edu.jlego.model.Package pkg = null;
  Imports imports = new Imports();
  Unit unit;
}
{
  [ pkg = PackageDeclaration() ]
  ( ImportDeclaration(imports) )*
  unit = TypeDeclaration(pkg, imports)
  <EOF>

  { return unit; }
}

org.swt.edu.jlego.model.Package PackageDeclaration() :
{
  String name;
}
{
  "package" name = Name() ";"

  { return new org.swt.edu.jlego.model.Package(name); }
}

void ImportDeclaration(Imports imports) :
{
  String name;
  StringBuffer buffer = new StringBuffer();
}
```

```
{
 "import" name = Name() { buffer.append(name); } [ "." { buffer.append("."); }
"*" { buffer.append("*"); } ] ";"

 { imports.add(new Import(buffer.toString())); }
}

Unit TypeDeclaration(org.swt.edu.jlego.model.Package pkg, Imports imports) :
{
 Unit unit = null;
}
{
 ( LOOKAHEAD( "component" )
  unit = ComponentDeclaration(pkg, imports)
 |
  LOOKAHEAD( "module" )
  unit = ModuleDeclaration(pkg, imports)
 |
  ";"
 )

 { return unit; }
}

Component ComponentDeclaration(org.swt.edu.jlego.model.Package pkg,
Imports imports) :
{
 Component comp = new Component();
 comp.setPackage(pkg);
 comp.setImports(imports);

 is_module = false;

 Token name;
 Composition composition;
}
{
 "component" name = <IDENTIFIER> { comp.setName(name.image); }
 [ "implements" composition = CompositeNameList() {
comp.setComposition(composition); } ]
 ComponentBody(comp)

 { return comp; }
}
```

```
Module ModuleDeclaration(org.swt.edu.jlego.model.Package pkg, Imports
imports) :
{
 Module module = new Module();
 module.setPackage(pkg);
 module.setImports(imports);

 is_module = true;

 Token name;
 Composition composition;
}
{
 "module" name = <IDENTIFIER> { module.setName(name.image); }
 [ "implements" composition = CompositeNameList() {
module.setComposition(composition); } ]
 ModuleBody(module)

 { return module; }
}

void ComponentBody(Component comp) :
{}
{
 "{" ( ComponentBodyDeclaration(comp) )* "}"
}

void ModuleBody(Module module) :
{}
{
 "{" ( ModuleBodyDeclaration(module) )* "}"
}

void ComponentBodyDeclaration(Component comp) :
{}
{
 LOOKAHEAD( "constructor" )
 ComponentConstructorDeclaration(comp)
|
 LOOKAHEAD( "method" )
 ComponentMethodDeclaration(comp)
|
 LOOKAHEAD( "module" )
 ModuleVariableDeclaration(comp)
|
 LOOKAHEAD( "component" )
```

```
 ComponentVariableDeclaration(comp)
|
 LOOKAHEAD( "invariants" "{" )
 InvariantsDeclaration(comp)
|
 LOOKAHEAD( "destructor" )
 ComponentDestructorDeclaration(comp)
}

void ModuleBodyDeclaration(Module module) :
{
}
{
 LOOKAHEAD( "constructor" )
 ModuleConstructorDeclaration(module)
|
 LOOKAHEAD( "method" )
 ModuleMethodDeclaration(module)
|
 LOOKAHEAD( "class" )
 ClassVariableDeclaration(module)
|
 LOOKAHEAD( "module" )
 ModuleVariableDeclaration(module)
|
 LOOKAHEAD( "destructor" )
 ModuleDestructorDeclaration(module)
}

void ModuleVariableDeclaration(Unit unit) :
{
 String type;
 Variable var;
}
{
 "module" type = Name() var = ModuleVariableDeclarator(type) {
unit.addVariable(var); }
 ( "," var = ModuleVariableDeclarator(type) { unit.addVariable(var); } )* ";"
}

Variable ModuleVariableDeclarator(String type) :
{
 Variable var = new Variable();
 var.setMeta("module");
 var.setType(type);
```

```
 Token token;
 String name;
}
{
  token = <IDENTIFIER> { var.setName(token.image); }
  [ "=" "load" { var.setAssignmentType("load"); } name = Name() {
var.setAssignmentName(name); }
FormalParameters(var.getAssignmentParameters()) ]

  { return var; }
}

void ComponentVariableDeclaration(Unit unit) :
{
 String type;
 Variable var;
}
{
 "component" type = Name() var = ComponentVariableDeclarator(type) {
unit.addVariable(var); }
 ( "," var = ComponentVariableDeclarator(type) { unit.addVariable(var); } )* ";"
}

Variable ComponentVariableDeclarator(String type) :
{
 Variable var = new Variable();
 var.setMeta("component");
 var.setType(type);

 Token token;
 String name;
}
{
 token = <IDENTIFIER> { var.setName(token.image); }
  [ "=" "load" { var.setAssignmentType("load"); } name = Name() {
var.setAssignmentName(name); }
FormalParameters(var.getAssignmentParameters()) ]

  { return var; }
}

void ClassVariableDeclaration(Unit unit) :
{
 String type;
 Variable var;
}
```

```
{
 "class" type = Name() var = ClassVariableDeclarator(type) {
unit.addVariable(var); }
 ( "," var = ClassVariableDeclarator(type) { unit.addVariable(var); } )* ";"
}

Variable ClassVariableDeclarator(String type) :
{
 Variable var = new Variable();
 var.setMeta("class");
 var.setType(type);

 Token token;
 String name;
}
{
 token = <IDENTIFIER> { var.setName(token.image); }
  [ "=" "new" { var.setAssignmentType("new"); } name = Name() {
var.setAssignmentName(name); }
FormalParameters(var.getAssignmentParameters()) ]

  { return var; }
}

void InvariantsDeclaration(Component comp) :
{
 Invariants invars = new Invariants();
 comp.setInvariants(invars);

 Invariant inv;
}
{
 "invariants" "{" ( inv = InvariantDeclaration() { invars.add(inv); } )* "}"
}

Invariant InvariantDeclaration() :
{
 Invariant inv = new Invariant();

 String exp;
 Token token;
 Token head;
}
{
 "invariant" "(" { head = getToken(1); } ConditionalExpression()
 {
```

```
   Variables vars = parseVariables(head, getToken(1));
   for(int i = 0; i < vars.getSize(); i++){
    inv.addVariable(vars.getVariable(i));
   }

   inv.setAssertion(traverse(head, getToken(1)).toString());
  }

  "," token = <STRING_LITERAL> { inv.setMessage(token.image); } ")" ";"

  { return inv; }
}

Condition ConditionDeclaration() :
{
 Condition cond = new Condition();

 String exp;
 Token token;
 Token head;
}
{
 "condition" "(" { head = getToken(1); } ConditionalExpression() {
cond.setAssertion(traverse(head, getToken(1)).toString()); }
 "," token = <STRING_LITERAL> { cond.setMessage(token.image); } ")" ";"

 { return cond; }
}

PreCondition PreconditionDeclaration() :
{
 PreCondition pre = new PreCondition();

 Condition cond;
}
{
 "pre" "{" ( cond = ConditionDeclaration() { pre.add(cond); } )* "}"

 { return pre; }
}

PostCondition PostconditionDeclaration() :
{
 PostCondition post = new PostCondition();

 Condition cond;
```

```
}
{
 "post" "{" ( cond = ConditionDeclaration() { post.add(cond); } )* "}"

 { return post; }
}

void ComponentMethodDeclaration(Component comp) :
{
 Method method = new Method();
 comp.addMethod(method);

 String type;
 Token token;
}
{
 "method" type = ResultType() { method.setReturnType(type); } token =
<IDENTIFIER> { method.setName(token.image); }
 FormalParameters(method.getParameters()) [ "throws"
ExceptionNameList(method.getExceptions()) ]
 "{" ComponentMethodBody(method) "}"
}

void ModuleMethodDeclaration(Module module) :
{
 Method method = new Method();
 module.addMethod(method);

 String type;
 Token token;
}
{
 "method" type = ResultType() { method.setReturnType(type); } token =
<IDENTIFIER> { method.setName(token.image); }
 FormalParameters(method.getParameters()) [ "throws"
ExceptionNameList(method.getExceptions()) ]
 "{" ModuleMethodBody(method) "}"
}

void ComponentMethodBody(Method method) :
{
 PreCondition pre;
 PostCondition post;
 Operation oper;
}
{
```

```
 [ LOOKAHEAD( "pre" "{" ) pre = PreconditionDeclaration() {
method.setPreCondition(pre); } ]
 [ LOOKAHEAD( "post" "{" ) post = PostconditionDeclaration() {
method.setPostCondition(post); } ]
 "operation" "{" oper = Operation() { method.setOperation(oper); } "}"
}

void ModuleMethodBody(Method method) :
{
 Operation oper;
}
{
 "operation" "{" oper = Operation() { method.setOperation(oper); } "}"
}

Operation Operation() :
{
 Operation operation = new Operation();
 Statement statement;
}
{
  ( statement = BlockStatement() { operation.addStatement(statement); } )*

 { return operation; }
}

void FormalParameters(Parameters params) :
{}
{
 "(" [ FormalParameter(params) ( "," FormalParameter(params) )* ] ")"
}

void FormalParameter(Parameters params) :
{
 Parameter param = new Parameter();
 params.add(param);

 String type;
 Token name;
}
{
 [ "final" ] type = Type() { param.setType(type); } token = <IDENTIFIER> {
param.setName(token.image); }
}

void ComponentConstructorDeclaration(Component comp) :
```

```
{
 Constructor con = new Constructor();
 comp.addConstructor(con);

 Token name;
}
{
 "constructor" FormalParameters(con.getParameters())
 [ "throws" ExceptionNameList(con.getExceptions()) ] "{"
ComponentConstructorBody(con) "}"
}

void ModuleConstructorDeclaration(Module module) :
{
 Constructor con = new Constructor();
 module.addConstructor(con);

 Token name;
}
{
 "constructor" FormalParameters(con.getParameters())
 [ "throws" ExceptionNameList(con.getExceptions()) ] "{"
ModuleConstructorBody(con) "}"
}

void ComponentConstructorBody(Constructor con) :
{
 PreCondition pre;
 PostCondition post;
 Operation oper;
}
{
 [ LOOKAHEAD( "pre" "{" ) pre = PreconditionDeclaration() {
con.setPreCondition(pre); } ]
 [ LOOKAHEAD( "post" "{" ) post = PostconditionDeclaration() {
con.setPostCondition(post); } ]
 "operation" "{" oper = ConstructorOperation() { con.setOperation(oper); } "}"
}

void ModuleConstructorBody(Constructor con) :
{
 Operation oper;
}
{
 "operation" "{" oper = ConstructorOperation() { con.setOperation(oper); } "}"
}
```

```
Operation ConstructorOperation() :
{
 Operation oper;
 ConstructorInvocation invo = null;
}
{
 [ LOOKAHEAD( "this" "(" ) invo = ExplicitConstructorInvocation() ]

 oper = Operation()

 {
  if(invo != null){
   oper.setConstructorInvocation(invo);
  }

  return oper;
 }
}

ConstructorInvocation ExplicitConstructorInvocation() :
{
 ConstructorInvocation invo = new ConstructorInvocation();

 Arguments args;
}
{
 "this" args = Arguments() ";" { invo.setArguments(args); }

 { return invo; }
}

void ComponentDestructorDeclaration(Component comp) :
{
 Destructor des = new Destructor();
 comp.setDestructor(des);
}
{
 "destructor" "{" ComponentDestructorBody(des) "}"
}

void ModuleDestructorDeclaration(Module module) :
{
 Destructor des = new Destructor();
 module.setDestructor(des);
}
```

```
{
 "destructor" "{" ModuleDestructorBody(des) "}"
}

void ComponentDestructorBody(Destructor des) :
{
 PreCondition pre;
 PostCondition post;
 Operation oper;
}
{
 [ LOOKAHEAD( "pre" "{" ) pre = PreconditionDeclaration() {
des.setPreCondition(pre); } ]
 [ LOOKAHEAD( "post" "{" ) post = PostconditionDeclaration() {
des.setPostCondition(post); } ]
 "operation" "{" oper = DestructorOperation() { des.setOperation(oper); } "}"
}

void ModuleDestructorBody(Destructor des) :
{
 Operation oper;
}
{
 "operation" "{" oper = DestructorOperation() { des.setOperation(oper); } "}"
}

Operation DestructorOperation() :
{
 Operation oper;
}
{
 oper = Operation()

 { return oper; }
}

/* Type, name and expression syntax */

String Type() :
{
 Token head;
}
{
 { head = getToken(1); } ( ( PrimitiveType() | Name()) ( "[" "]" )* )

 { return traverse(head, getToken(1)).toString().trim(); }
```

```
}

void PrimitiveType() :
{}
{
  "boolean"
|
  "char"
|
  "byte"
|
  "short"
|
  "int"
|
  "long"
|
  "float"
|
  "double"
}

String ResultType() :
{
  String result;
}
{
  (
    "void" { result = "void"; }
  |
    result = Type()
  )

  { return result; }
}

String Name() :
/*
 * A lookahead of 2 is required below since "Name" can be followed
 * by a ".*" when used in the context of an "ImportDeclaration".
 */
{
  StringBuffer buffer = new StringBuffer(16);
  Token token;
}
{
```

```
  ( token = <IDENTIFIER> { buffer.append(token.image); } | "result" {
buffer.append("result"); } )
  ( LOOKAHEAD(2) "." token = <IDENTIFIER> {
buffer.append('.').append(token.image); } )*

  { return buffer.toString(); }
}

Exceptions ExceptionNameList(Exceptions exps) :
{
  String name;
}
{
  name = Name() { exps.add(new org.swt.edu.jlego.model.Exception(name)); } (
"," name = Name() { exps.add(new org.swt.edu.jlego.model.Exception(name)); }
)*

  { return exps; }
}

Composition CompositeNameList() :
{
  Composition comp = new Composition();
  String name;
}
{
  name = Name() { comp.add(new Composite(name)); } ( "," name = Name() {
comp.add(new Composite(name)); } )*

  { return comp; }
}

/*
 * Expression syntax follows.
 */

void Expression() :
/*
 * This expansion has been written this way instead of:
 *   Assignment() | ConditionalExpression()
 * for performance reasons.
 * However, it is a weakening of the grammar for it allows the LHS of
 * assignments to be any conditional expression whereas it can only be
 * a primary expression.  Consider adding a semantic predicate to work
 * around this.
 */
```

```
{
}
{
 ConditionalExpression()
 [
  AssignmentOperator() Expression()
 ]
}

void AssignmentOperator() :
{
 is_assignment_statement = true;
}
{
 "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | ">>>=" | "&=" | "^=" | "|="
}

void ConditionalExpression() :
{}
{
 ConditionalOrExpression() [ "?" Expression() ":" ConditionalExpression() ]
}

void ConditionalOrExpression() :
{}
{
 ConditionalAndExpression() ( "||" ConditionalAndExpression() )*
}

void ConditionalAndExpression() :
{}
{
 InclusiveOrExpression() ( "&&" InclusiveOrExpression() )*
}

void InclusiveOrExpression() :
{}
{
 ExclusiveOrExpression() ( "|" ExclusiveOrExpression() )*
}

void ExclusiveOrExpression() :
{}
{
 AndExpression() ( "^" AndExpression() )*
}
```

```
void AndExpression() :
{}
{
 EqualityExpression() ( "&" EqualityExpression() )*
}

void EqualityExpression() :
{}
{
 InstanceOfExpression() ( ( "==" | "!=" ) InstanceOfExpression() )*
}

void InstanceOfExpression() :
{}
{
 RelationalExpression() [ "instanceof" Type() ]
}

void RelationalExpression() :
{}
{
 ShiftExpression() ( ( "<" | ">" | "<=" | ">=" ) ShiftExpression() )*
}

void ShiftExpression() :
{}
{
 AdditiveExpression() ( ( "<<" | ">>" | ">>>" ) AdditiveExpression() )*
}

void AdditiveExpression() :
{}
{
 MultiplicativeExpression() ( ( "+" | "-" ) MultiplicativeExpression() )*
}

void MultiplicativeExpression() :
{}
{
 UnaryExpression() ( ( "*" | "/" | "%" ) UnaryExpression() )*
}

void UnaryExpression() :
{}
{
```

```
  ( "+" | "-" ) UnaryExpression()
|
  PreIncrementExpression()
|
  PreDecrementExpression()
|
  UnaryExpressionNotPlusMinus()
}

void PreIncrementExpression() :
{}
{
  "++" PrimaryExpression()
}

void PreDecrementExpression() :
{}
{
  "--" PrimaryExpression()
}

void UnaryExpressionNotPlusMinus() :
{}
{
  ( "~" | "!" ) UnaryExpression()
|
  LOOKAHEAD( CastLookahead() )
  CastExpression()
|
  PostfixExpression()
}

// This production is to determine lookahead only.  The LOOKAHEAD
specifications
// below are not used, but they are there just to indicate that we know about
// this.
void CastLookahead() :
{}
{
  LOOKAHEAD(2)
  "(" PrimitiveType()
|
  LOOKAHEAD("(" Name() "[")
  "(" Name() "[" "]"
|
  "(" Name() ")" ( "~" | "!" | "(" | <IDENTIFIER> | "this" | Literal() )
```

```
}

void PostfixExpression() :
{}
{
 PrimaryExpression() [ "++" | "--" ]
}

void CastExpression() :
{}
{
 LOOKAHEAD("(" PrimitiveType())
 "(" Type() ")" UnaryExpression()
|
 "(" Type() ")" UnaryExpressionNotPlusMinus()
}

void PrimaryExpression() :
{}
{
 PrimaryPrefix() ( LOOKAHEAD(2) PrimarySuffix() )*
}

void PrimaryPrefix() :
{}
{
 Literal()
| LOOKAHEAD( "this" "." )
 "this"
|
 "(" Expression() ")"
|
 AllocationExpression()
|
 LOOKAHEAD( ResultType() "." "class" )
 ResultType() "." "class"
|
 Name()
}

void PrimarySuffix() :
{}
{
 LOOKAHEAD(2)
 "." "this"
|
```

```
 LOOKAHEAD(2)
 "." AllocationExpression()
|
 "[" Expression() "]"
|
 "." <IDENTIFIER>
|
 Arguments()
}

void Literal() :
{}
{
 <INTEGER_LITERAL>
|
 <FLOATING_POINT_LITERAL>
|
 <CHARACTER_LITERAL>
|
 <STRING_LITERAL>
|
 BooleanLiteral()
|
 NullLiteral()
}

void BooleanLiteral() :
{}
{
 "true"
|
 "false"
}

void NullLiteral() :
{}
{
 "null"
}

Arguments Arguments() :
{
 Arguments args = new Arguments();
}
{
 "(" [ ArgumentList(args) ] ")"
```

```
  { return args; }
}

void ArgumentList(Arguments args) :
{
  Argument arg;
  Token head;
}
{
  { head = getToken(1); } Expression() { args.add(new Argument(traverse(head,
getToken(1)).toString())); }
  ( "," { head = getToken(1); } Expression() { args.add(new
Argument(traverse(head, getToken(1)).toString())); } )*
}

void AllocationExpression() :
{
}
{
  LOOKAHEAD( "load" )
  LoadAllocationExpression()
|
  NewAllocationExpression()
}

void NewAllocationExpression() :
{}
{
  { if(!is_module) throw new ParseException(); }
  { is_new_statement = true; }

  "new" Name() Arguments()
}

void LoadAllocationExpression() :
{}
{
  { is_load_statement = true; }
  "load" Name() Arguments()
}

/*
 * Statement syntax follows.
 */
```

```
Statement Statement() :
{
 Token head = getToken(1);
 Statement statement;
}
{
 { is_new_statement = is_load_statement = is_assignment_statement = false; }
 (
   LOOKAHEAD(2)
   LabeledStatement()
 |
   Block()
 |
   EmptyStatement()
 |
   StatementExpression() ";"
 |
   SwitchStatement()
 |
   IfStatement()
 |
   WhileStatement()
 |
   DoStatement()
 |
   ForStatement()
 |
   BreakStatement()
 |
   ContinueStatement()
 |
   statement = ReturnStatement() { return statement; }
 |
   ThrowStatement()
 |
   SynchronizedStatement()
 |
   TryStatement()
 )

 {
  if(is_new_statement){
    return new Statement.NewStatement(traverse(head, getToken(1)).toString());
  }
  else if(is_load_statement){
    return new Statement.LoadStatement(traverse(head, getToken(1)).toString());
```

```
    }
    else if(is_assignment_statement){
      return new Statement.AssignmentStatement(traverse(head,
getToken(1)).toString());
    }
    else{
      return new Statement.NormalStatement(traverse(head,
getToken(1)).toString());
    }
  }
}

void LabeledStatement() :
{}
{
  <IDENTIFIER> ":" Statement()
}

void Block() :
{}
{
  "{" ( BlockStatement() )* "}"
}

Statement BlockStatement() :
{
  Token head;
  Statement statement;
}
{
  LOOKAHEAD([ "final" ] Type() <IDENTIFIER>)
  { head = getToken(1); } LocalVariableDeclaration() ";" { return new
Statement.NormalStatement(traverse(head, getToken(1)).toString()); }
|
  statement = Statement() { return statement; }
}

void VariableDeclarator() :
{}
{
  <IDENTIFIER> [ "=" Expression() ]
}

void LocalVariableDeclaration() :
{}
{
```

```
  [ "final" ] Type() VariableDeclarator() ( "," VariableDeclarator() )*
}

void EmptyStatement() :
{}
{
  ";"
}

void StatementExpression() :
/*
 * The last expansion of this production accepts more than the legal
 * Java expansions for StatementExpression.  This expansion does not
 * use PostfixExpression for performance reasons.
 */
{
}
{
  PreIncrementExpression()
|
  PreDecrementExpression()
|
  PrimaryExpression() [ "++" | "--" | AssignmentOperator() Expression() ]
}

void SwitchStatement() :
{}
{
  "switch" "(" Expression() ")" "{"
    ( SwitchLabel() ( BlockStatement() )* )*
  "}"
}

void SwitchLabel() :
{}
{
  "case" Expression() ":"
|
  "default" ":"
}

void IfStatement() :
/*
 * The disambiguating algorithm of JavaCC automatically binds dangling
 * else's to the innermost if statement.  The LOOKAHEAD specification
 * is to tell JavaCC that we know what we are doing.
```

```
 */
{}
{
 "if" "(" Expression() ")" Statement() [ LOOKAHEAD(1) "else" Statement() ]
}

void WhileStatement() :
{}
{
 "while" "(" Expression() ")" Statement()
}

void DoStatement() :
{}
{
 "do" Statement() "while" "(" Expression() ")" ";"
}

void ForStatement() :
{}
{
 "for" "(" [ ForInit() ] ";" [ Expression() ] ";" [ ForUpdate() ] ")" Statement()
}

void ForInit() :
{}
{
 LOOKAHEAD( [ "final" ] Type() <IDENTIFIER> )
 LocalVariableDeclaration()
|
 StatementExpressionList()
}

void StatementExpressionList() :
{}
{
 StatementExpression() ( "," StatementExpression() )*
}

void ForUpdate() :
{}
{
 StatementExpressionList()
}

void BreakStatement() :
```

```
{}
{
 "break" [ <IDENTIFIER> ] ";"
}

void ContinueStatement() :
{}
{
 "continue" [ <IDENTIFIER> ] ";"
}

Statement ReturnStatement() :
{
 Token head;
}
{
 "return" { head = getToken(1); } [ Expression() ] ";" { return new
Statement.ReturnStatement(traverse(head, getToken(1)).toString()); }
}

void ThrowStatement() :
{}
{
 "throw" Expression() ";"
}

void SynchronizedStatement() :
{}
{
 "synchronized" "(" Expression() ")" Block()
}

void TryStatement() :
/*
 * Semantic check required here to make sure that at least one
 * finally/catch is present.
 */
{}
{
 "try" Block()
 ( "catch" "(" FormalParameter(null) ")" Block() )*
 [ "finally" Block() ]
}
```

WORKS CITED

Edwards, S. (1997). Representation Inheritance: A Safe Form of "White Box" Code Inheritance. *IEEE Transactions On Software Engineering*, Volume 23(2).

Frakes, W. B., & Pole, T. P. (1994). An Emperical Study of Representation Methods for Reusable Software Components. *IEEE Transactions On Software Engineering*, Volume 20(8).

Harms, D. (1991). Copying and Swapping: Influences on the Design of Reusable Software Components. *IEEE Transactions On Software Engineering*, Volume 17(5).

Khan, E. H., Al-A'ali, M., & Girgus, M. R. (1995). Object-Oriented Programming for Structured Procedural Programmers. *IEEE Computing Practices*, Volume 18(48).

Kiziltan, Z., Jonsson, T., & Hnich, B. (2000). On the Definition of Concepts in Component Based Software Development. *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications.*

Liu, Y., & Cunningham, C. (2002). Software Component Specification Using Design by Contract. *Southeastern Software Engineering Conference.*

Merriam-Webster. (1989). *Webster's Dictionary of English Usage.*

McDirmid, S., Flatt, M., & Hsieh, W. C. (2001). Jiazzi: New-Age Components for Old-Fashioned Java. *Conference on Object-Oriented Programming, Systems, Languages, and Applications.*

Munson, John. (2003). Software Engineering Measurement. New York, NY: Auerbach Publications.

Parnas, H. (1972). On The Criteria To Be Used On Decomposing Systems Into Modules. *Communications of the ACM*, Volume 12(12).

Sullivan, K. J., Marchukov, M., & Socha, J. (1999). Analysis of a Conflict Between Aggregation and Interface Negotiation in Microsoft's Component Object Model. *IEEE Transactions On Software Engineering*, Volume 25(4).

Ward, P. T. (1989). How to Integrate Object Orientation with Structured Analysis and Design. *IEEE Software Development Concepts*, Volume 7(74).

Wohlin, C. and Runeson, P. (1994). Certification of Software Components. *IEEE Transactions On Software Engineering*, Volume 20(6).

VITA

Micah Spears was born in Beaumont, Texas, on May 2, 1980, the son of Clyde and Sharon Spears. After completing his work at Lumberton High School, he went on to Lamar University where he studied Computer Science and received his Bachelor of Science in December 2001. He moved to San Marcos in August 2002 and began graduate studies in Computer Science at Texas State University-San Marcos.

Permanent Address:

25 Candlewick
Lumberton, Texas 77657

This thesis was typed by Micah Spears.