International Conference on Information and Communication Technologies (ICICT 2014)

Foundational UML Behavioral Specification with Java

Renu George\textsuperscript{a,}\textsuperscript{*}, Philip Samuel\textsuperscript{b,}\textsuperscript{*}

\textsuperscript{a}College of Engineering, Chengannur, 689101, India

\textsuperscript{b}Cochin University of Science and Technology, Kochi, 682022, India

Abstract

An executable UML model has a detailed behavioural specification that enables it to run as a program. The detailed specification enables to test and validate the model independent of any implementation platform. The foundational UML (fUML) specification adopted in 2008 provided the first operational base semantics of activity modelling. Since the fUML specification did not provide any new concrete syntax, in order to execute the model one had to draw a very detailed activity diagram. This led to the development of a textual representation to specify the computations which led to the development of action language for foundational UML known as Alf. The UML behaviour specified using textual notation in Alf can be attached to a UML model at any place. Syntactically, Alf looks like C, C++ or Java and semantically it maps to the fUML subset. Alf provides an additional layer of abstraction. In this paper we demonstrate how Java can be used in place of Alf to specify detailed behaviour. Java is a popular high level language among developers and useful to the industry. Specifying UML behaviour with Java avoids the extra burden of developers studying an unfamiliar new abstraction termed Alf.

Keywords: fUML; Alf; executable UML; lexical structure; parsing; UML behavior specification;

1. Introduction

Graphical modeling languages are applied in system design and documentation. A widely used general purpose graphical modeling language is Unified Modeling Language (UML). Previously, the dynamic aspects of the system
such as method bodies and exit/entry actions of state machines could not be described with UML, and natural language or an existing programming language had to be used to describe the dynamic aspects. The drawbacks of using these methods are 1) natural language is ambiguous and early system simulation and model checking may be affected by ambiguous descriptions 2) programming languages are usually implementation specific. To systematically build realistic behaviour, precise action semantics has been incorporated into UML 2.0. UML can be made executable by making it more expressive in describing the dynamic aspects of the system.

A model is an abstraction of a system. It is hard to achieve consistency at the modeling level due to the abstraction gap between modeling and programming languages. Due to this gap native code need not be aware of the concepts defined in models. Models provide an abstraction of the system by omitting unnecessary details. Actions, algorithms and navigation paths cannot be described in detail by UML diagrams. Rules and constraints for building the model has to be described using some programming language or textual representation. Model aware action languages are developed to reduce the abstraction gap. The Action Language for Foundational UML (Alf) was defined by the Object Management Group (OMG) to reduce the gap. Models are primarily represented using graphical notations of UML. To describe the elements of the model and to specify executable behaviour, a UML action language is used. Alf acts as the surface notation for specifying executable behaviours within the model. A complete specification of behaviour and computation is required to specify full executability of the system. Using a textual notation it is easy to add detailed behaviour to models and to run models as programs thereby specifying full executability. The language is referred to as the Action Language for Foundational UML, Alf.

In this paper we demonstrate that Java can be used to specify UML behaviour by comparing the lexical and syntactic constructs of Alf and Java. We show that the syntax of Alf closely resembles the syntax of Java language and the lexical constituents are also almost similar with a few exceptions. The compilation process in Alf maps the Alf behavioural specification either to a fUML model or to an executable code on a non UML platform whereas the compilation of a Java program produces an intermediate platform independent representation, the byte code.

2. Related Work

There are many papers related to the translational execution of Alf and the generation of code from design models. The paper, mapping UML associations into Java code, describes the implementation of binary associations in Java considering multiplicity, navigability and visibility. Although there are no direct constructs in Java to directly implement binary associations, by combining classes, attributes and methods, associations can be implemented.

Paper provides an investigation and evaluation of UML action languages. In this paper, a technology called Umple is defined that allows any language to be used as an action language. A higher level of abstraction is provided by Umple when compared with the existing high level languages. Paper discusses how Java code can be generated from state machines using design patterns. In this paper, each state is considered as an object. Each state in the state chart becomes a class, and all the transitions and actions of the state are encapsulated in it. To implement hierarchical and concurrent states, the concepts of composition and delegation can be used.

Paper discusses about how to bridge the chasm between executable metamodeling and models of computation. A new framework to combine a metamodel and Models of Computation (MoC) in a modular fashion is introduced in this paper. A complete and executable definition of a Domain specification language (DSL) is provided, along with how a given MoC can be reused for different domain-specific metamodels, and how different MoCs can be used for a given metamodel to account for variants of a DSL. In, Alf language is used to perform a data flow analysis of models. The paper also describes that precise data flow information can be found by mapping UML state machines to Alf.
3. Java

Java is one of the most popular and widely used object-oriented languages. Java is a platform independent language in the sense that the code produced by Java can be run on a variety of CPUs under different environments. The design, testing and refinement of Java were performed by real working programmers. Hence the language has grounds on the experience of real programmers. Java is logically consistent and cohesive. Programmer has full control over the language. Java language also has influence on the internet. To provide network security, Java creates a firewall between computer and the networked application. The output of Java compiler is byte code which is not executable. It is interpreted on a Java Virtual Machine that can run on any platform. By compiling Java code to byte code, cross-platform programs can be created. The advantages of Java programming language are it is simple, secure, object oriented, robust, multi-threaded, architecture-neutral, interpreted, dynamic and distributed.

Class diagram is a UML diagram that represents the static aspects of the system. Association and generalization relationships exist between classes. Java is an object-oriented programming language, hence the concept of classes, objects and inheritance are supported by Java. Classes can be directly implemented in Java. Generalization relationships in class diagram can be modeled using the inheritance concept in Java. Association represents the relationship between classes and it is one of the key concepts of UML. Associations cannot be directly represented by Java but they can be implemented either by using a combination of classes, attributes and methods or by mapping to Java interface. Single associations can be implemented using an attribute corresponding to the link in the target class; multiple associations can be implemented using Java collections such as Vector, HashSet etc. Navigability includes unidirectional and bidirectional associations and by including attributes in the corresponding classes, navigability can be implemented. The visibility levels of Java and UML have the same semantics; hence visibility can also be implemented in Java.

Java language is provided with a variety of features that makes it suitable for representing the elements of UML models. Dynamic or behavioural modeling is done with activity diagrams and state chart diagrams. Activity diagrams are a specialized form of state chart diagrams and they depict the procedural flow of actions. Design patterns can be used to generate Java code from state machines. During the translation of states to code, each state is considered as an object of the class that embeds all transitions and actions. Concurrent states can also be represented in Java using composition and delegation.

The features of Java enable it to specify all the constructs of UML either directly or by combining a set of features. Hence Java can be used to specify the behaviour associated with the elements of the UML models.

4. Alf

The Action Language for Foundational UML (Alf) was developed to provide a textual representation for UML modeling elements. The Alf concrete syntax can be mapped to the abstract syntax of the standard Foundational Subset for Executable UML Models (known as Foundational UML or fUML). This mapping specifies the execution semantics of Alf and the result of executing an Alf text is determined by the semantics of the fUML model to which it is mapped.

Models are represented using the graphical notations of UML and executable behaviours are specified using Alf which serves as the surface notation for specifying executable behaviours of UML elements. This is the primary goal of Alf. Specification of behaviour includes methods on the operations of classes or transition effect behaviours on state machines. The extended notation provided with Alf can be used to represent structural modeling elements. The Alf syntax only directly covers the fUML subset which is a subset of UML and the entire model represented using fUML subset can be specified with Alf.

The design of Alf is based on the following guiding principles.
There is a close resemblance between the syntax of Alf and C or Java. UML textual syntax like double colon for name qualification can also be used in Alf.

It is possible to embed Alf text in the graphical models without any change in the model. (e.g., including special characters in names, using constructors with arbitrary names etc.). The ease of accommodating Alf in graphical models is useful 1) in providing the execution semantics during mapping to the fUML subset and 2) during mapping to other non fUML models.

To reference elements outside of an activity, Alf uses the naming system provided by UML namespaces. Local names are provided to reference flows of values within an activity.

Alf uses an implicit type system and it also uses static type checking based at least on typing declared in the structural model elements.

Alf has the expressivity of Object Constraint Language (OCL) in the use and manipulation of sequences of values. These sequence expressions are fully executable in terms of fUML expansion regions; Alf also allows simple and natural specification of highly concurrent computations.

To be an action language is the primary goal of Alf and within the bounds of the fUML subset, Alf also provides concrete syntax for structural modeling.

5. Comparison of the syntax of Java and Alf

A programming language is specified in terms of its lexical constituents, syntactic specification and semantic specification. The lexical structure specifies the building blocks of a programming language. It specifies how individual characters are grouped together to form tokens. The aim of syntactic specification is to group the tokens to form syntactically correct structures. A structural description of the various expressions that form legal sentences in the language is specified using syntax. The syntax of the statements is specified using a grammar. Semantics specifies the meaning of syntactically valid statements in the language. An Extended Backus-Naur Form (EBNF) is used to specify the lexical and syntactic structure of Alf action language. The Alf concrete syntax can be mapped to the abstract syntax of the standard fUML as defined in the fUML specification. The semantics of the mapped fUML model determines the execution semantics and the result of executing the Alf text. The syntax of Java programming language is specified using context free grammar. The compilation of a java program generates a set of class files for each class in the source program. The comparison of Alf and Java languages are based on 14,15 and 18.

5.1. Lexical Structure

A programming language has a lexical structure that determines how sequence of characters are grouped together to form tokens. The lexical structure of Alf defines how the sequence of characters in an Alf input text is divided into a set of input elements or tokens. Such input elements can be categorized as whitespace, comments or tokens. Tokens are again categorized as documentation comment, name, reserved word, literal, punctuation and operator.

Java statements also have a lexical structure consisting of white space, comments and tokens. The tokens can be identifier, keyword, literal, separator or operator. The lexical structure of Alf language resembles the lexical structure of Java language in that the lexical analyzer detects white space, comments and tokens. The tokens in Alf and Java are almost similar with a few exceptions such as documentation comments, names (almost similar to identifiers) etc.

5.2 Data Types

Alf supports the following primitive data types – Boolean, String, Integer, natural and bit string. Java language also supports the data types byte, short, char, int, long, float, double, Boolean. All the primitive data types supported by Alf are also supported by Java programming language.
5.3 Class

A classifier whose instances are objects is called a class. A class may include properties and operations. Properties define the attributes of the class and they can have the values of the appropriate type. Operations in a class can be invoked by objects of the same class using invocation expressions. Properties and operations resemble the attributes and methods of a Java class. Java and Alf supports classes and both have the syntax to specify abstract and concrete classes. Table.1 illustrate examples of abstract and concrete class definitions in both Alf and Java. The examples demonstrate the close resemblance in class definitions in Alf and Java.

Table 1. Examples of abstract and concrete class definition in Alf and Java.

| Alf                          | Java                      |
|------------------------------|---------------------------|
| Abstract Class               | Abstract class Point      |
| and abstract                 | translate{                |
| Operation                    | int x = 2, y = 3;        |
|                              | void move (int dx, intdy) |
|                              | x += dx;                 |
|                              | y += dy;                 |
|                              | alert();                 |
|                              | abstract void alert();   |
|                              | }                         |
| Concrete class               | class product {           |
|                              | private qty: Count;      |
|                              | public getQuantity():    |
|                              | Count {                  |
|                              | return self.quantity;    |
|                              | }                        |
|                              | class product {           |
|                              | private int qty;         |
|                              | public int getQty (){    |
|                              | return qty;              |
|                              | }                        |

5.4 Statements

Statements do not have values and are segments of behaviour that are executed for their effect. The primary unit of sequencing and control in the Alf representation of behaviour is a statement. A statement sequence is an Alf text consisting of a set of linearly ordered statements. To specify UML behaviour, such statement sequences may be attached to UML models. For larger syntactic constructs a block is used which is a delineation of a statement sequence.

Alf action language supports annotated, inline, block, empty, local name declaration, expression statements, if, switch, while, do, for, break, return, accept and classify statements. All the statements in a statement sequence are executed sequentially in order. Using an opaque behaviour, an Alf statement sequence can be inserted into a UML model or UML activity. The body of the opaque behaviour is specified using the unprocessed text of the Alf statement sequence and the corresponding language string is Alf. An action may have input and output pins and it should be named. The names of the input pins are considered to be the assigned sources within the statement sequence. The output pins may be assigned names within the statement sequence; they provide the values for the corresponding output pin at the end of the statement sequence. Within the statement sequence, the names of other visible model elements (qualified as necessary) may also be used as usual.

In Java, statements control the sequence of execution and do not have values. Similar to Alf, they are executed for their effect. A sequence of statements constitutes a block. In Java, a statement can be an if - else statement, assert statement, while, do - while, for, break, continue, return, throw, switch, try catch statements. The syntax of the statements in Alf closely resembles Java syntax. As a comparison, the syntax of while and switch statements in Alf and Java are compared in Table.2. The examples in Table.2 outline the close resemblance of Alf to Java. The only difference in the switch statement is in the use of break statements in Java.
Table 2. Comparison of switch and while statements.

| Type of Statement | Alf | Java |
|-------------------|-----|------|
| While statement   | "while" "(" conditionExpression ")" statement | while (ConditionalExpression) Statement |
| Example of while statement | While (file.hasNext()) { Record = file.readNxt(); while (file.hasNext()) { record = readNxt(); } | |
| Switch statement  | "switch" "(" Label ")" "{" case exp ":" exp statement "default" ";" Default statements "}" | switch (Label) { case Exp : Exp statement break; default : Default statement break; } |
| Example of switch statement to check the validity of a date | switch (month) { case 1: case 3: case 5: case 7: case 8: case 10: case 12: if (day>0 && day<=31) Writeln("Valid date"); case 4: case 6: case 9: case 11: if (day>0 && day<=30) Writeln("Valid date"); case 2: if ( ((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0)) { if (day>0 && day<=29) Writeln("Valid date"); } else { if (day>0 && day<=28) Writeln("Valid date"); } default: writeLine("Invalid date"); } | switch (month) { case 1: case 3: case 5: case 7: case 8: case 10: case 12: if (day>0 && day<=31) System.out.println("Valid date"); break; case 4: case 6: case 9: case 11: if (day>0 && day<=30) System.out.println("Valid date"); break; case 2: if ( ((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0)) { if (day>0 && day<=29) System.out.println("Valid date"); } else { if (day>0 && day<=28) System.out.println("Valid date"); } break; default: System.out.println("Invalid date"); break; } |

5.5 Expressions

An expression evaluates to a collection of values and is treated as behavioural unit. The side effects of evaluating an expression are changing the value of an attribute of an object. Using an opaque expression, an Alf expression can be inserted into a UML model. To specify values in the UML model, an Alf expression may be used. Alf supports both conditional expression and assignment expression. The Java language expression syntax, also supports both conditional expression and assignment expression. The formal specification of the execution semantics of an Alf expression are given by the mapping to UML activity graphs. Since Java and Alf supports conditional and assignment expressions, Alf expressions can be replaced with expressions in Java.

6. Parsing

Parsing is the process of verifying the syntactic correctness of the input program or text. Parsing is a process that takes as input a stream of tokens, verifies the syntax and produces as output a parse tree or syntax tree. The syntax is
usually defined using a grammar. A syntactic grammar whose definition is based on extended Backus-Naur form (EBNF) is used to specify the Alf concrete syntax. The Alf concrete syntax defines how an abstract syntax tree is formed by grouping lexical tokens. This is done by grouping the tokens produced by the lexical analysis of an Alf text to construct an abstract syntax tree. A hierarchical parse tree is produced as the result of parsing. The parsing of an Alf input text is thus essentially the same as is done for the processing of any typical textual programming language. Usually a context free grammar is used to specify the syntax of programming languages. The use of EBNF grammar provides context free parsing of Alf text. The atomic expressions from statements and the specified UML action associated with these expressions are identified by the parser. The specified unit is parsed into an abstract syntax representation in memory. There is no pre-compilation of units and any unit imported should be recursively compiled. During parsing, violations of the abstract syntax constraints defined in the Alf specification are checked and reported. Static semantic analysis is performed to gather additional information for mapping to fUML whereas in the case of programming language, semantic analysis is performed for type checking and resolving names. The difference in the semantic analysis phase arises due to the difference in the output produced after compilation. In the case of Alf, it is mapped to fUML, whereas in Java, a set of class files is produced.

The final step in processing an Alf text is the mapping to a representation in terms of fUML abstract syntax. By performing a depth first traversal of the annotated abstract syntax tree with derived attributes from static semantic analysis, the fUML abstract syntax representation is built. The fUML reference implementation maps all parsed units to the in-memory representation. Unit mapping is executed at an execution environment created at a fUML locus. In the case of Java, after the compilation is performed, a platform independent representation called bytecode is produced. Bytecodes are stored in a class file.

A set of Java source files are compiled to a set of class files by the Java compiler. The lexical analyzer groups characters in the source file to form tokens. The tokens are then grouped by the parser to generate the abstract syntax tree. A symbol table with symbol definitions is created. Semantic analysis is performed after performing process annotations on the abstract syntax tree. It includes name resolution, type checking and constant folding. After performing data flow analysis of the annotated parse tree, inner classes, class literals etc. are removed and a class file for each class is generated. The compilation steps in both Alf and Java are almost similar except in the output produced by the compiler.

7. Conclusion

Alf is a textual representation of a modeling language for a subset of UML called foundational UML. It has been shown that Alf has syntax similar to Java programming language. Java is a high level platform independent programming language. Executable behaviors within a UML model can be specified with Alf. The Alf code can be mapped to fUML models or high level language code after compilation whereas in Java, a set of class files containing a platform independent representation called bytecode is produced which can be interpreted and executed. After parsing, an intermediate representation: fUML model or high level language code in the case of Alf and bytecode in the case of Java is produced. Alf produces a layer of abstraction between the models and code. In this paper we have shown that since Alf resembles Java, in many of the constructs, the extra level of abstraction can be avoided using a high level programming language such as Java. Although using an action language to provide the behavioral specification helps in the generation of full code, extra effort is put in to produce the textual representation in Alf code and further map the models to fUML and then later execute the models. Alf syntax is similar to C, C++ or Java and the programmer has to learn another language, Alf to make the models executable. Java is one of the most popular high programming languages and it has been demonstrated that many of the Alf constructs are similar to Java. The advantages of using Java to specify behaviors within the UML model are that it eliminates an extra layer of abstraction and programmer is relieved from the burden of learning another language Alf which is similar in many aspects to high level programming language.

References
1. Object Management Group (OMG): Unified Modeling Language: Superstructure, version 2.0 available at http://www.omg.org
2. Yang, Mikai, Greg J. Michaelson, and Rob Pooley. Formal Action Semantics for a UML Action Language. *J. UCS* 14.21 2008: 3608-3624
3. Ciccozzi, Federico, Antonio Cicchetti, and Mikael Sjodin. Towards Translational Execution of Action Language for Foundational UML. In *Software Engineering and Advanced Applications (SEAA) 2013 39th EUROMICRO Conference*: IEEE, 2013. p. 153-160.
4. Mellor, S.J., Tockey, S. Action Semantics for UML, Response to OMG RFP ad/98-11-01 OMG ad/1002-08-04; *Tech. Rep.* Project Technology, Inc.
5. Erni, D., & Kuhn, A. The Hacker’s Guide to javac. University of Bern, *Bachelor’s thesis, supplementary documentation*, 2008.
6. Ed Seidewitz. Programming in UML: An Introduction to fUML and Alf, *Tutorial for the OMG Executable UML Information Day 22 March 2011*
7. Object Management Group Action Language for Foundational UML (Alf), *Concrete Syntax for a UML Action Language*, beta 1 version OMG Document Number.ptc/2010-10-05. 2010
8. Gonzalo Génova, Carlos Ruiz Del Castillo, and Juan Llorens. Mapping UML associations into Java code. *Journal of Object Technology* 2.5 2003: 135-162.
9. Badreddin, Omar Bahy, Timothy C. Lethbridge, and Andrew Forward. Investigation and Evaluation of UML Action Languages. In *MODELSWARD2014*, p. 264-273.
10. Michel, Daniel, and Andreas Steffen. Code Generator for UML State Machines. 2011.
11. Combemale, B., Hardebolle, C., Jacquet, C., Boulanger, F., &Baudry, B. Bridging the chasm between executable metamodeling and models of computation. In *Software Language Engineering Springer Berlin Heidelberg*: 2013. p. 184-203.
12. Obaid, S., S. Asghar, and M. Naeem. Data Flow Analysis of UML Models by ALF. *International Journal of Trade, Economics & Finance* 5.1; 2014
13. Harrison, William, Charles Barton, and MukundRaghavachari. Mapping UML designs to Java. ACM Vol. 35. No. 10. 2000.
14. Action Language for Foundational UML (Alf), *Concrete Syntax for a UML Action Language*, 23 August 2010, Available at http://www.omg.org/spec/ALF/1.0.1/PDF/
15. James Gosling, Bill Joy, Guy Steele, GiladBracha, *The Java™ Language Specification*. Third Edition, Addison-Wesley, May 2005
16. Herbert Schildt. *The Complete reference Java™ 2*. Fifth Edition. Tata McGraw-Hill;2002
17. Slonneger, Kenneth, and Barry L. Kurtz. *Formal syntax and semantics of programming languages*. Reading: Addison-Wesley, 1995.
18. Gosling, James, ed. *The Java language specification*. Addison-Wesley Professional, 2000.