

# Proof Transforming Compilation of Eiffel Contracts

## DIPLOMA PROJECT PLAN

Project period: 7. January – 6. May 2008  
Student name: Hasan KARAHAN  
Status: 10<sup>th</sup> Semester  
Email address: hkarahan@student.ethz.ch  
Supervisor name: Martin Nordio

### 1. PROJECT DESCRIPTION

#### Overview

A proof-transforming compiler is a compiler that takes a source proof as input and produces a bytecode proof. This project consists of the development of a proof-transforming compiler for Eiffel contracts. The compiler takes a source proof with contracts and produces a bytecode proof and its CIL contract. The proof translation will be done using the compiler implemented in [4].

#### Scope of the work

The work consists of the implementation of a contract translation. The input of the compiler is a source proof and contracts written in the XML format. In a first step, a parser for the source proof will be implemented. The parser will produce an AST. The design of the XML schema and the AST is presented in [4]. They will be extended to handle contracts.

In a second step, contracts will be parsed generating an AST. The AST will be designed to model the complete contract [3]. The formal grammar of the considered contract language is the following:

```
datatype EiffelContract = Requires_ensures boolExpr  
datatype boolExpr = Const bool  
| Neg boolExpr  
| And boolExpr boolExpr  
| Or boolExpr boolExpr  
| AndThen boolExpr boolExpr  
| OrElse boolExpr boolExpr  
| Xor boolExpr boolExpr  
| Impl boolExpr boolExpr  
| Eq expr expr  
| NotEq expr expr  
| Less expr expr  
| Greater expr expr  
| LessE expr expr  
| GreaterE expr expr  
| Type typeFunc  
datatype typeFunc = ConformsTo typeExpr typeExpr  
| IsEqual typeExpr typeExpr  
| IsNotEqual typeExpr typeExpr
```

<b>datatype</b> <i>typeExpr</i>	=	<b>EType</b> <i>EiffelType</i>
		<b>Type</b> <i>expr</i>
<b>datatype</b> <i>expr</i>	=	<b>ConstInt</b> <i>int</i>
		<b>RefVar</b> <i>varID</i>
		<b>Attr</b> <i>objID attribID</i>
		<b>CallR</b> <i>callRoutine</i>
		<b>Create</b> <i>EiffelType routine argument</i>
		<b>Old</b> <i>expr</i>
		<b>Bool</b> <i>boolExpr</i>
		<b>Void</b>
<b>datatype</b> <i>callRoutine</i>	=	<b>Call</b> <i>expr routine argument</i>
<b>datatype</b> <i>argument</i>	=	<b>Argument</b> <i>expr</i>

*EiffelTypes* are *Boolean*, *Integer*, classes with a *classID* or *None*. The notation (*cID* : *classID*) means, given an Eiffel class *c*, *cID(c)* returns its *classID*.

<b>datatype</b> <i>EiffelType</i>	=	<b>Boolean</b>
		<b>Integer</b>
		<b>EClass</b> ( <i>cID</i> : <i>classID</i> )
		<b>None</b>

Variables are local variables or parameters, *Result*, or *Current*:

<b>datatype</b> <i>var</i>	=	<b>Var</b> <i>vID EiffelType</i>
		<b>Result</b> <i>EiffelType</i>
		<b>Current</b> <i>EiffelType</i>

Attributes are defined with a variable *ID* and an *EiffelType*. Routines can take only one argument. Routines are defined with a routine *ID*, the argument type and the return type.

<b>datatype</b> <i>attrib</i>	=	<b>Attr</b> ( <i>aID</i> : <i>attribID</i> ) <i>EiffelType</i>
<b>datatype</b> <i>routine</i>	=	<b>Routine</b> <i>routineID EiffelType EiffelType</i>

Later, the contract parser will be extended to parse a super set of the contract language which includes universal and existential quantifiers. This parser will be used to generate an AST from the source pre- and postconditions.

Finally, a contract translator will be implemented. This module will take the AST of the contract language and produce the CIL contract in First Order Logic. The formalization of this translation is presented in [3].

Thus, the compiler will parse the following rules:

- assignment, conditional, compositional instructions
- loops, rescue rule, remove
- retry instruction by variable
- check instruction
- objects: creation; read, write attributes; invocation
- routine implementation; subtype rule; local rule
- once procedure; once functions
- language-independent rules:
  - false axiom
  - strength and weak
  - invariant
  - substitution
  - conjunction and disjunction
  - all-rule and ex-rule

The subset of the Eiffel language is the following:

```
exp          ::= literal | var | exp op exp
stm          ::= x := exp | stm; stm | from stm until exp loop stm end
              | if exp then stm else stm end
              | check exp end
              | create {Type} routine_name ( exp )
              | x := y.Type@a
              | y.Type@a := exp
              | x := y.Type : routine_name ( exp )
once_routine ::= name (var : Type) : Type is
                require boolExpr
                [ local var : Type, ... ]
                once stm
                [ rescue stm ]
                ensure boolExpr
                end
non_once_routine ::= name (var : Type) : Type is
                  require boolExpr
                  [ local var : Type, ... ]
                  do stm
                  [ rescue stm ]
                  ensure boolExpr
                  end
routine       ::= once_routine | non_once_routine
```

### *Intended results*

The result will be a proof-transforming compiler for a subset of Eiffel and it consists of:

1. A parser for the source proof.
2. A parser for the contract language.
3. A parser for a super set of the contract language.
4. A contract translator.

## **2. BACKGROUND MATERIAL**

### *Reading & Reference list*

- Gobo Eiffel Tools and Libraries: <http://www.gobosoft.com>
- The Expat XML Parser: <http://expat.sourceforge.net>
- eXML - Eiffel XML Parser Toolkit: <http://efsa.sourceforge.net/archive/leitner/exml.htm>

## **3. PROJECT MANAGEMENT**

### *Objectives and priorities*

The priority lies in the implementation of the XML parser; it should produce the AST corresponding the rules mentioned in section 1.

### *Criteria for success*

A working parser which can process a given proof in XML. If the input is not correct, the compiler will show an error message indicating the line where the error was found. The project will succeed if at least the parser for the source proof, the parser for the contract language and the contract translator is implemented. Test classes will be provided. The implementation will follow the Eiffel methodology where the classes will include contracts and at least 70% of the routines will provide their contracts.

## 4. PLAN WITH MILESTONES

### *Project steps*

1. Implementation of a parser for source proofs given in XML.
2. Implementation of a sub-parser for contracts which is able to deal with boolean expressions.
3. Implementation of sub-parser for post- and preconditions which is able to deal with boolean expressions (*including* quantifiers).
4. Implementation of a contract translator.
5. Integration of the results into EiffelStudio.

## REFERENCES

- [1] Chair of Software Engineering: *Semester-/Diplomarbeiten*; Online at: <http://se.inf.ethz.ch/projects/index.html>, consulted in January 2008.
- [2] Bertrand Meyer: *Object-Oriented Software Construction, 2nd edition*, Prentice Hall, 1997.
- [3] Martin Nordio, Peter Müller, and Bertrand Meyer: *Formalizing Proof-Transforming Compilation of Eiffel programs*, ETH Zurich, January 2008.
- [4] Michel Guex: *Implementing a Proof-Transforming Compiler from Eiffel to CIL*, Semester Thesis, ETH Zurich, 2006.
- [5] Peter Müller and Martin Nordio: *Proof-transforming compilation of programs with abrupt termination*, SAVCBS '07: Proceedings of the 2007 conference on Specification and verification of component-based systems, ACM, 2007.

WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to
1	Preparation	Jan 7	Jan 11	5d	5d		1,600	hkarahan
2	<b>XML Processing</b>	<b>Jan 14</b>	<b>Feb 1</b>	<b>15d</b>	<b>15d</b>		<b>4,800</b>	
2.1	Parser Implementation	Jan 14	Jan 18	5d	5d		1,600	hkarahan
2.2	AST Adaptation	Jan 21	Feb 1	10d	10d		3,200	hkarahan
3	<b>Boolean Expressions</b>	<b>Feb 4</b>	<b>Feb 22</b>	<b>15d</b>	<b>15d</b>		<b>4,800</b>	
3.1	Parser Implementation	Feb 4	Feb 8	5d	5d		1,600	hkarahan
3.2	AST Adaptation	Feb 11	Feb 22	10d	10d		3,200	hkarahan
4	<b>P&amp;P Conditions</b>	<b>Feb 25</b>	<b>Mar 7</b>	<b>10d</b>	<b>10d</b>		<b>3,200</b>	
4.1	Parser Implementation	Feb 25	Feb 29	5d	5d		1,600	hkarahan
4.2	AST Adaptation	Mar 3	Mar 7	5d	5d		1,600	hkarahan
5	Contract Translator	Mar 10	Mar 21	10d	10d		3,200	hkarahan
6	Eiffel Studio Integration	Mar 24	Apr 4	10d	10d		3,200	hkarahan
7	<b>Quality Assurance</b>	<b>Apr 7</b>	<b>Apr 11</b>	<b>5d</b>	<b>5d</b>		<b>1,600</b>	
7.1	Testing	Apr 7	Apr 9	3d	3d		960	hkarahan
7.2	Debugging	Apr 10	Apr 11	2d	2d		640	hkarahan
8	Documentation	Apr 14	May 2	15d	15d		4,800	hkarahan
9	Presentation	May 5	May 9	5d	5d		1,600	hkarahan

