Code Generation of Ontology-based abstract Algorithms

Liang Shan

A dissertation submitted to the
University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science

Submitted September 2008

DECLARATION

I, declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Liang Shan
7th September 2008

PERMISSION TO LEND AND/OR COPY

Liang Shan
7th September 2008

Acknowledgements

I would like to thank many people for helping me during my M.Sc. research project. First, and foremost, I would like to deeply thank Trinity PHD Simone Grassi for his guidance, words of encouragement, kindness and patience. I shall never forget his support in the past months.

I also would like to take the opportunity to thank my supervisor, Stephen Barrett, for his guidance, unstoppable enthusiasm and support throughout the project.

I would also like to express my appreciation to all my NDS classmates who provided me with caring and inspiration during difficult times.

Finally, I am so very grateful for the family I am blessed with. To my Mum Yan HuiHua and Dad Shan GengBao, who put a lifetime of effort and love into my upbringing. Thank you for everything you have done for me.

Abstract

Software Development as part of software engineering is a huge and always active field of Computer Science. The traditional waterfall software development life cycle shows its weakness in providing user oriented service since it is hard to gather all the requirements and changes may happened any time after deployment. Especially, with the development of internet, there is a strong increase in the need of more dynamic, pluggable service for better interaction between services and easy maintenance. One of solutions is the use of Model Driven techniques. A modeling technique can be used to express information, knowledge or systems in a structure that is defined by a consistent set of rules. Rules are used to interpret the meaning of components in the structure.

Trinity PHD Simone Grassi's has carried out his research to create an abstract specification of algorithms (based on a set of ontologies) as a Model Driven Platform to build software. The set of ontologies (OWL) are created to host a rich set of semantic information to the modeling algorithm and this approach decouples the algorithm from any particular architecture, framework or programming language. The most two important ontology files of Simone's abstract modeling algorithm are Algorithm Ontology (AO) and Specific System Logic Ontology (SSLO). AO is tree structure model contains individual that constitute an algorithm and SSLO is for a specific language include syntax rule for languages features (control flow, operators, variable, etc) using XSLT.

This project is a collaboration that delivers a platform in support of Simone's research. This project report presents the design and implementation of a Code Generation Engine that the CGE interprets AO using SSLO to generate code. The generated code through CGE can be function, full classes, or any valid code and then the code can be deployed on the server as online service. CGE is very generic and not dependent on any programming language/framework and it is easy to extend to a new language by creating a new SSLO. Code can be modified and regenerated by changing the AO at demand, making the service dynamic and pluggable.

Test cases were designed and tested to ensure the initial requirements of CGE are met. CGE meets Simone's research aim that it is not to generate any possible programming language structure or trick, but only to generate valid code starting from the ontology that is the model of an abstract algorithm. CGE is a successful tool for the testing of specification of abstract algorithms and it will be core part for test and evaluation part of a paper to be submitted in the future.

Table of Contents

LIST OF FIGURES

Chapter 1 Introduction

1.1 Background

Software development is a huge and always active field of Computer Science. Many research projects and technologies proposal continually appear in this field in order to gain high productivity, portability, interoperability and easy maintenance. During the years, more and more importance was given to the maintenance. Systems are becoming larger and more complex, open to other technologies both from the use of libraries based on code and for the interaction during execution time with other systems.

The concept of service became an accepted word to identify a processing making available a computation to remote processes on the internet. In particular web service is one of the most widely used technologies, it enables remotes system to interact for access data, request computation and communicate the result of an executed process or ask for a process to be executed [1]. Those services on the internet make the maintenance and update much more complicate. Traditional waterfall software development life cycle shows its weakness in providing user oriented online service since it is hard to gather all the requirements at first. Changes may happen at any time after service deployed. So there is a huge growth of demands for dynamic, pluggable service and easy maintenance.

1.2 Motivation

One of solutions for providing dynamic, flexible service is the use of Model Driven techniques. A modeling technique can be used to express information, knowledge or systems in a structure that is defined by a consistent set of rules. Rules are used to interpret the meaning of components in the structure.

Trinity PhD student Simone Grassi has carried out his research to create an abstract specification of algorithms as a Model Driven Platform to build software. His approach is based on ontologies and allows decoupling the algorithm specification from the system that will host service based on these algorithms. This project start as a branch of overall research project to provide solution to generated code (PHP, Java) using a language independent ontology based abstract algorithm.

1.3 Objectives

The main aim of this project is to design and implement a Code Generation Engine that take ontology-based modeling algorithm as input and generate code for specific language as output.

The following goals are derived

- The CGE is completely independent from any specific system or programming language, in such a way that generation of code for different language is a matter of changing the input of the CGE and not the CGE itself in any part of it.
- Generated code can be deployed on the web server and accessed through internet
- Code can be modified and regenerated by changing the model of the algorithm on demanded, there is no changes should be made to CGE.
- The CGE can generate code for a new programming language by creating a new language specific ontology algorithm, without changing the CGE itself.

1.4 Project Approach

I started in studying ontology-based modeling algorithm approach to derive lists of requirements for CGE and also researched in other related topic such as Model Driven Architecture, Semantic Web. The aims of this project are clearly derived and pointed.
The state of art in code generation was researched and key processes involved in development of a code generator were identified and I then did a few practices for get familiar with technologies (Protégé, Saxon API) to used in the project implementation CGE is designed in a specific architecture so that it makes CGE to be decoupled from any particular programming language or systems. Various functions were designed to handle different processes involving in code generation.
CGE is then implemented in Java using various third party APIs to achieve code generation of ontology-based abstract algorithms and code generated using CGE can then be deployed on the web server and run as online service. Meantime, I also worked together with Simone in fulfilling language specific ontology by writing XSL for language features.
Test cases were designed and tested to evaluate whether CGE meets the initial requirements.

1.5 Contribution

CGE is very generic and it is not dependent on any programming language or framework but only depends on what modeling algorithms describe. It is very easy to supporting a new language by creating a new language specific ontology.
Code generated by the CGE can be modified at user demand by changing the model of the algorithm. Any change need to be done to any part of the CGE, only the input given to the CGE change.

CGE project meets Simone's research aim that it is not to generate any possible programming structure or trick, but only to generate valid code starting from the ontology that is the model of an abstract algorithm.
CGE is a successful tool for the testing of specification of abstract algorithms and it will be a core part test and evaluation part of a paper to be submitted in the future.
By using of CGE, it provides ability for supporting more dynamic and pluggable web service. User can ask for new version of web service any time by make modification to ontology algorithm and CGE can automatically derive new version of service and make it redeployed.

1.6 Roadmap

Chapter 2 will describe the background information on the areas of Model Driven, Semantic Web which they are related to this dissertation. These areas were the starting point for all research, and a presented here to familiarize the reader with concepts upon which the project is based.

Chapter 3 will describe an overview in the current state of the art of code generation, the benefit of using code generator and advantage of code generation use ontology over tradition XML based code generation.

Chapter 4 will describe in detail about the ontology-based modeling algorithm approach and key owl files construct the algorithm.

Chapter5 will describes requirements in relation to code generation engine are listed. And the introduction of technologies to be used in implementation is also included. Following the design for XSL style sheet used in the language specific ontology and detail design for code generation engine.

Chapter 6 will describe implementation of language specific ontology for both PHP and Java language features. And then it is concentration on explanation in detail explanation of how CGE was implemented by developing various classes and functions for different purposes and how they are interacted to make CGE work.

Chapter7 will evaluate the CGE based on multiple test cases. Algorithm ontology file were created to be used as input for CGE for specific test purposes. All those AO files are used to evaluate whether CGE implemented to meet the initial aim. In this chapter, also include explanation in details about how to make change to AO using Protégé.

Chapter 8 will describes project summary, influence from collaboration work and lists of contributions achieved and point out future work to be done in the further development.

Chapter 2 Background

This chapter provides background information on the areas of problem of traditiona software development, Model Driven Aritecture, Semantic Web which they are related to this dissertation. These areas were the starting point for all research, and a presented here to familiarize the reader with concepts upon which the project is based.

2. 1 Problems of Traditional Software Development

- **Productivity Problem:**

As we can see from the following classical software development cycle shown in Figure 2-1, that it usually break down into 5 phases as Requirement Analysis, Design, Implementation, Testing, Maintenance[2]. Normally requirement analysis and system design phrase will consume most of development to produce large numbers of diagrams such as system class diagrams, state diagrams and sequence diagrams and documents related to system design. Until implementation stage is reach, it will result in actual code producing, then those produced code will be run through set of test cases to ensure they meet the system requirement. Finally tested system will be deployed to available to users.

However, it is very common that changes are needed to be made during testing or after deployment, which these changes will bring a another new round of software development cycle that new UML will be added and some of original UML diagrams need to modify and redraw. Design document needed to be reedited and produced again. Then software engineer is going to make actual changes to the code and test code again. Changes can still happen at any time if user is not happy about system and there is no guarantee of zero bug in the system yet. The productivity is quite low if changes to system happen a lot that precious time of software engineer will be cost in rewriting design documentation and redraw UML diagram instead of programming.

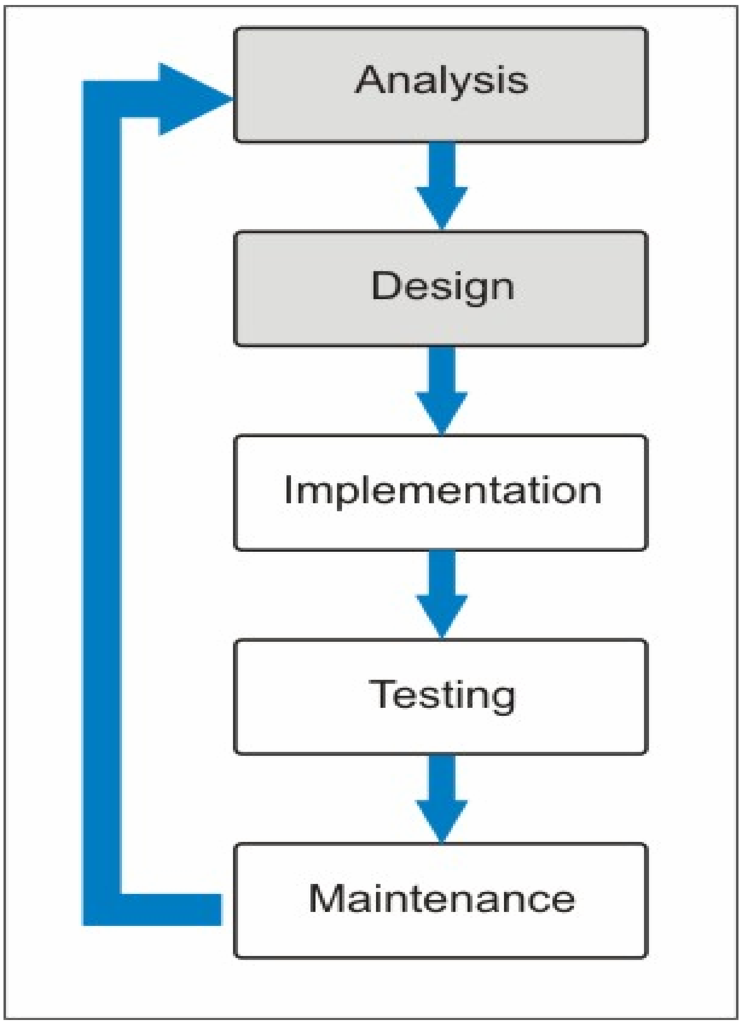


Figure 2-1: Traditional Software Development Life Cycle

- **Portability Problem**

Software industry has its unique character which is rapid and continuous update of technologies[4]. New generation technologies are brought to industry ever quicker than before that makes software industry different from most other industries.

The demand for portable system grows because many company managers realized the value of applying new technologies will bring to their companies. For example XML, J2EE, .Net platform are widely used to develop enterprise software application, those technologies are already proved their power in enabling business and vendors like Microsoft, IBM, Sun are keeping promoting their new technologies.

People want to jump on these new technologies, but problems arise in most cases they have to give up current system developed in old technologies or make significant changes to original system to adapt new technologies. Lack of portability will result in huge cost in system redevelopment and as a consequence it is not welcomed.

- **Interoperability Problem**

In ancient age of software industry, software systems were intended to be customer specific and isolated from other systems[5]. However, since the internet become more and more important, the need for interaction between software systems is growing. At the moment, most software system can be divided into several components for different functions. Example like web based application can be partitioned into three layers as User Interaction Layer, Business Logic Layer and Persistent Layer and there are a lot of small components inside each layer to perform their functions [6]. Those small components of entire web application need to interact with to perform business logic so that user change data in the web page can result in the update of data in the back end database.

On the other hand, it is very common that there are more than one technology involved in the system development. For web based application example mentioned before that JSP and Servlet may be used for Front Tier to user and it also need use relational databases as a storage mechanism. Building System based on components bring a lot flexibility for interaction between systems. This also helps make it easier to make changes to a system. System built based on set of technologies, also need to be interacted with each other.

- **Maintenance Problem**

As we already mentioned that changes can cause big pain in the traditional software development cycle. However other problems emerge in the maintenance especially after system deployment that most of companies need to hire internal staffs or pay for external consultants to be responsible for maintain system regularly at daily basis and also responsible for new service deployment if necessary. One tiny change made to system will result in extra workload for not only software engineer but also to system administrator.

If changes made to system under permit of manager can automatically generate code (new service or modification to original) which it will save a lot of time in service redevelopment and redeployment. But authentication and security control should be concerned at this point since no manager wants to see his system can differentiate without control.

2.2 Model Driven Architecture

MDA is own and trademark by the Object Management Group since 2000 and MDA is an approach to using model techniques in driving software development.

The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationship of different kind of models and how to use sets of model to derive software applications [7].

Within Model Driven Architecture, we can use the following definition for Model

- A Model is a description of (part of) a system written in a well-defined language as shown in Figure 2-2.
- A Well-defined language is a language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer [3]

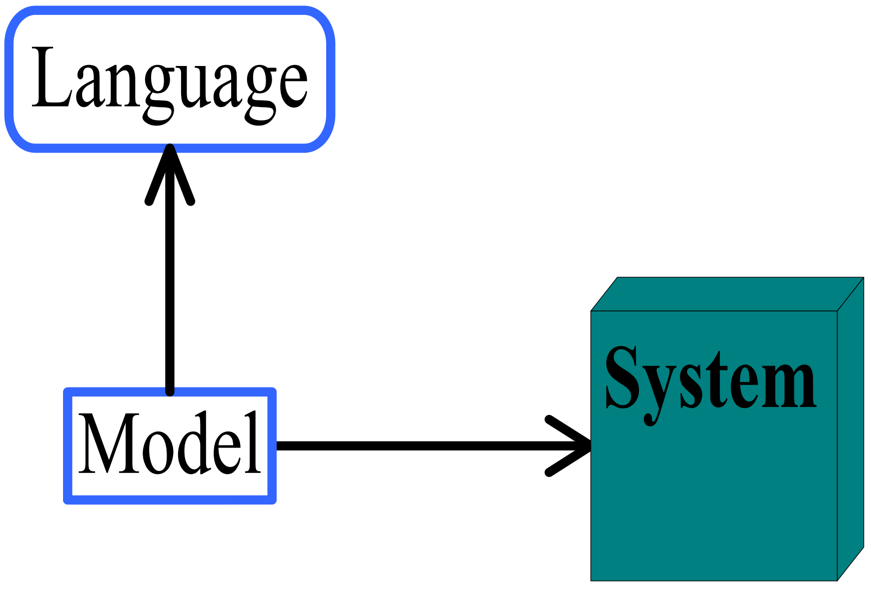


Figure 2-2: Model Definitions

2.2.1 Model Driven Architecture Development Life Cycle

The MDA development life cycle which is shown in Figure 2-3, looks similar to the traditional life cycle. The same stages are identified. Major difference form traditional software development cycle is the use of Platform Independent Model and Platform Specific Model in MDA development.

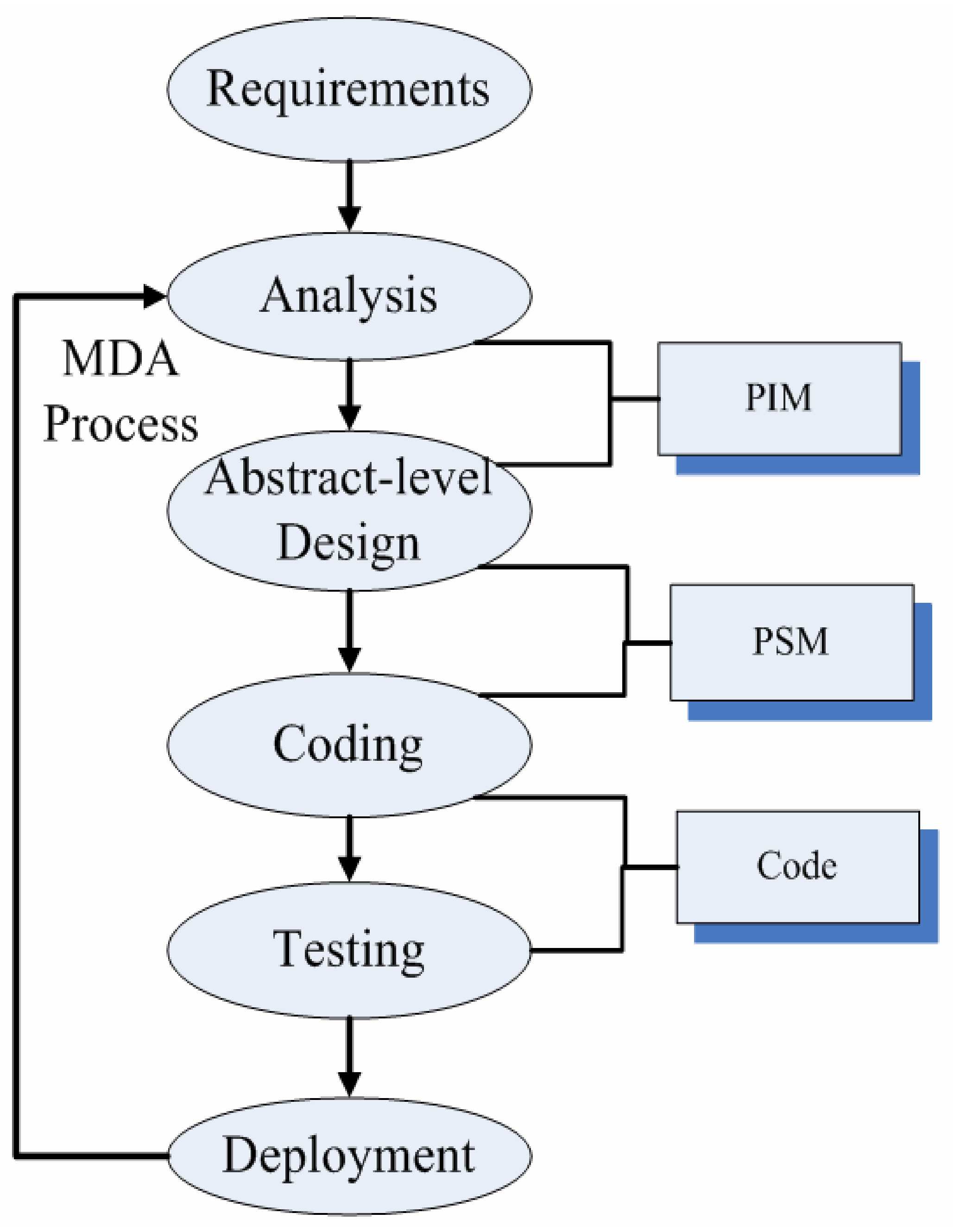


Figure 2-3: MDA Software Development Life Cycle

- **Platform Independent Model (PIM)**

PIM describes an abstract model for system. PIM describe a system without any knowledge of final implementation from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type [8]. PIM decouple itself from the concrete implementation and any kind of technologies to be used in the future.

- **Platform Specific Model (PSM)**

A platform specific model describes a system with full knowledge of final implementation of final implementation platform from platform specific viewpoint. Platform here can be specific technologies concepts such as Java, C++, C# or different system architecture. A PSM combines the specification in the PIM with the details that specify how that system uses a particular type of platform [2].

- **PIM to PSM to Code Automatic Transformation**

The PIM is able to transform into one or more PSM. For each specific technology that one PSM is going to be generated. The abstraction of PIM makes it very flexible to switch between the technologies by extending a PSM. And if the system covers more than one technology, then there will be many PSMs with one PIM. One thing has to be mention that transformation should be automatically by meaning of using transformation tools or programs. More and more tools and programs are developed to help either PIM to PSM or PSM to Code transformation. Finally PSM is transformed into Code following by certain predefined rules. Because PSM is designate to specific technology so that major concern is about code generation in the correct order and non syntax-error style.

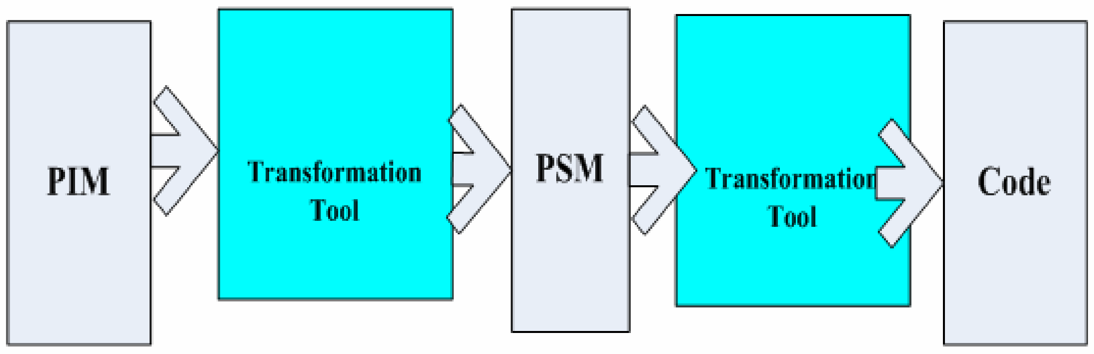


Figure 2-4: Major Steps in MDA Development Processes

2.3 Ontology and OWL

2.3.1 Ontology

The emergence of the Semantic web (Berners-Lee 1999) has caused a growing need for knowledge reuse, and has strengthened its potential at the same time. Therefore, ontologies and problem-solving methods (which in some cases are considered as the precursors of Semantic Web Services) are playing an important role in this context. Ontologies used to represent reusable and sharable pieces of domain knowledge and how they can be used in applications[9]. In this context, ontologies are reusable and sharable artifacts that have to be developed in a machine interpretable language. Ontology is a formal, explicit specification of a shared conceptualization. (Studer Benjamins, & Fensel 1998) [10].

Common Components of Ontology

- **Class** represents concepts which it depends on which domain it in. For instance, in the traveling domain, concepts are: locations (cities, village, ect.), lodging (hotel, camping, etc.) and means of transport (plain, trains, cars, ferries, motorbikes and ships). Classes in ontology are usually organized in taxonomies through which inheritance mechanisms can be applied.
- **Relation** represents a type of association between concepts of the domain. They are formally defined as any subset of a product of n sets. [11] Ontology usually contains binary relations. The first argument is known as the domain of the relation, and the second argument is the range. Relations can also be instantiated with knowledge from the domain.
- **Attributes** are usually distinguished from relations because their range is a datatype, such as string, number and so forth, while the range of relations is a concept.
- **Axiom** is a sentence in first order logic that is assumed to be true without proof. In practice, we use axioms to refer to the sentences that cannot be represented using only slots and values on a frame [12].

2.3.2 OWL

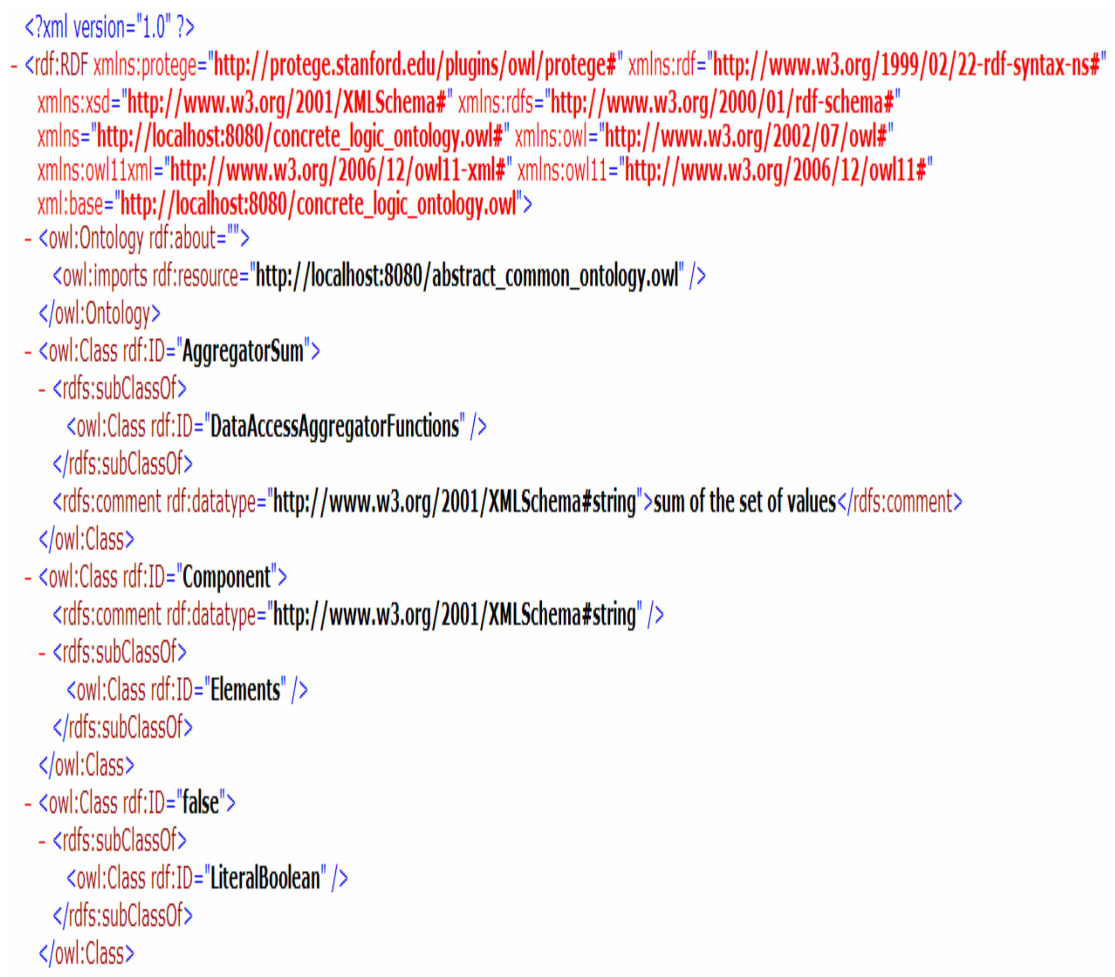
OWL stands for Web Ontology Language is a W3C standard, Developed from its predecssors OIL (Fensel, Horrocks, van Harmelen, McGuinness, & Patel-Schneider,

s2001) and DAML + OIL (Patel-Schneider, Horrocks, & van Harmelen, 2002), it is at present the standard ontology languge on the web[13]. The data describe by an OWL ontology is interprets as set of individuals and a set of property assertion which relate these individuals to each other.
W3C OWL specification includes the definition of three variant of OLW, with different levels of expressive.

OWL Lite: The least expressive of the OWL, Compared with RDFS it adds local range restrictions, existential restriction, simple cardinality restriction, equality, and various types of property[13].

OWL DL: By comparing with OWL Lite was design to provide the maximum expressiveness possible while retaining computational completeness (all entailments are guaranteed to be computed) and decidability (all computation will finish in finite time) of reasoning systems [14]. It was designed to support the existing business logic and computational properties for reasoning systems.

OWL Full: is designated for user who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees [14].



```
  <?xml version="1.0" ?>
- <rdf:RDF xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://localhost:8080/concrete_logic_ontology.owl#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:owl11xml="http://www.w3.org/2006/12/owl11-xml#" xmlns:owl11="http://www.w3.org/2006/12/owl11#"
    xml:base="http://localhost:8080/concrete_logic_ontology.owl">
  - <owl:Ontology rdf:about="">
      <owl:imports rdf:resource="http://localhost:8080/abstract_common_ontology.owl" />
    </owl:Ontology>
  - <owl:Class rdf:ID="AggregatorSum">
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="DataAccessAggregatorFunctions" />
      </rdfs:subClassOf>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">sum of the set of values</rdfs:comment>
    </owl:Class>
  - <owl:Class rdf:ID="Component">
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="Elements" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:ID="false">
    - <rdfs:subClassOf>
        <owl:Class rdf:ID="LiteralBoolean" />
      </rdfs:subClassOf>
    </owl:Class>
```

Figure 2-5: OWL Sample

Chapter 3 State of Art

This chapter will give an overview in the current state of the art of the benefit of using code generator, XML based code generation, and advantage of code generation use ontology over tradition XML based code generation.

3.1 Benefits of using Code Generator

- **Change tolerance**

This principle describes level of system maintainability and reusability. For handwritten programs, it depends on either designed system structure is generic or not. It there is high coupling between components in a system which it makes change the existing structure extreme difficult and will bring a lot of pain to redevelopment. Code generation is designated to handle change made to abstract model.

Code Generation shows its advantage in change tolerance. Because the final output not depends on the structure of Code Generator but depends on the abstract model. Because the abstract model is designated for modeling the structure of system in a flexible way, change should be easily make to the abstract model and finally differentiate the output from Code Generator.

- **Correctness**

For human being to produce high quality code with less syntax errors or bugs inside, it depends on software engineers' experience, and other quality assurance processes such as review, testing.

Correctness from code generator point of view is shifted to the design of system rather than the generated code because if code generator works fine as designed which output code should be correct[22]. On the other hand, code generator provides a convenient way for people who are not familiar specific language to produce code instead make it hand-coded.

- **Performance**

Software engineers try to minimize required resources, such as data, code, network traffic, and CPU time. Improving performance usually increase efficiency of data processing which may involve system structure refinement.

For code generator, performance depends on how efficient generator is able to process initial source file (owl algorithm file in this project) and generate required code such as Java or PHP. Chose appropriate API for owl file parsing is very important as I found that processing owl algorithm file consume most of time by comparing with output programming code generation.

- **Language switched flexibility**

It is very difficult for any software engineers to have knowledge for all kinds of programming languages. The emergence of code generator show its great advantage in providing a flexible way of modify abstract model (XML, UML) to derive concrete implementation, so user of code generator only need to have knowledge about how to modify the abstract model and code generator will interprets abstract model to code.

3.2 XML based Code Generator

XML is a standard representation for information. It can be used to create customize information structure for any domain or system. In XML based code generator that it is used as the syntax for representing programming specification form which to generate code.

In this XML approach, once the input document is defined in XML, XSLT scripts can be written to process XML documents and generate output documents in various forms. A number of free XSLT processors are available on the internet such as Saxon.

Key feature of a XML based Code Generator are:

- **Parser:** XML document parsing in implemented in the XSLT processor, otherwise with a customer parser, one need to implement the parser or understand how to use a program generator such as lex/yacc to generate the parsing framework [15].
- **Tree processing:** The XSLT processor provides access to the XML through the XPATH expression and provides many programmatic constructions and functions to perform the tree processing. On the other hand, XSLT users can write XSLT scripts to perform operations on the tree. XSLT programming for code generation is at a high level which is at the level of tree abstraction[16]. With the use of XSLT, code generation programming is at the tree level abstraction, the programmer never needs to worry about tree data structure implementation details.
- **Writer:** The XSLT processor implements this function as well.

Figure 3-1 shows the process of document generation using XSL and XSLT. Location of each file should be identified as part of the generation.

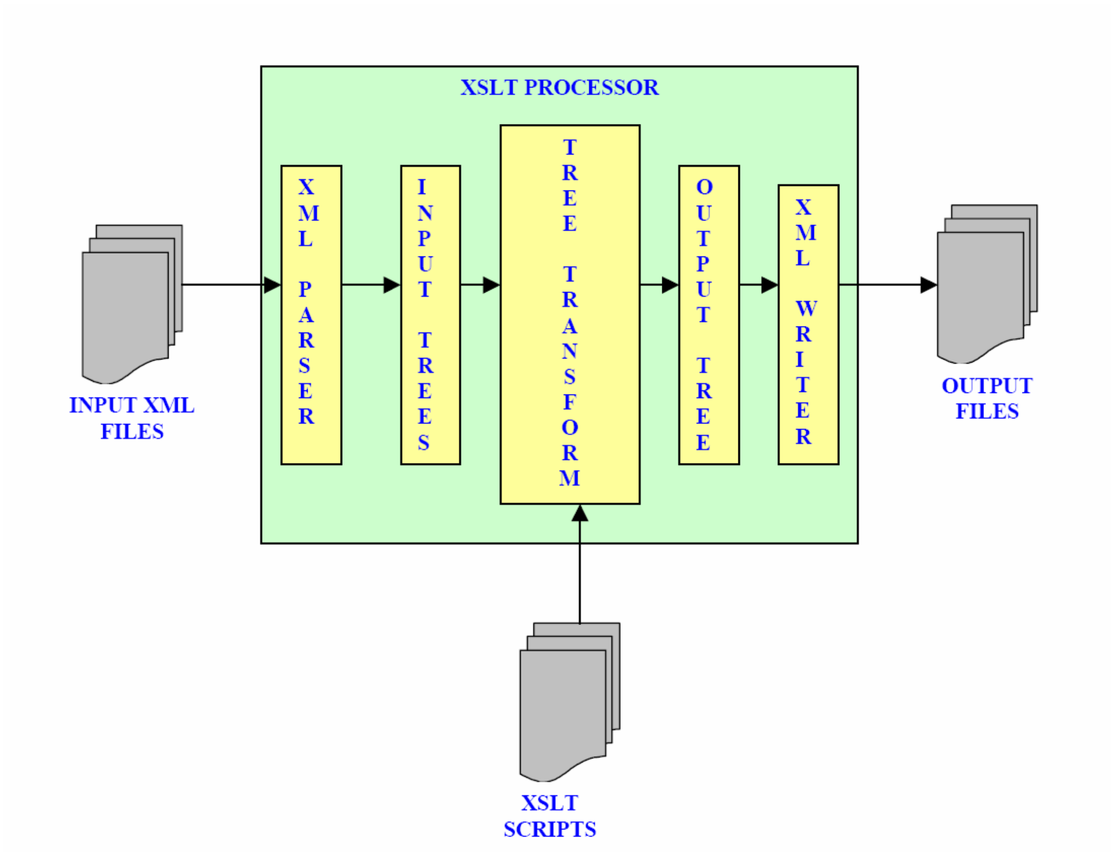


Figure 3-1: XML based Code Generation Processes

3.3 Advantages of Ontology-based Code Generation over XML-based Code Generation

As we introduced before that XML is mainly used to store data and represent data structure however XML structure can be defined by anyone for any purpose there is no any knowledge of semantic information can be retrieved from XML.

Ontology describes knowledge based on logic. Standard like OWL specify semantic information used to infer additional information using reasoning and there are an availability of tools for OWL (Protégé). By using ontology-based modeling algorithm approach can add a rich set of semantic information.

Chapter 4 Ontology-based algorithm modeling approach

This chapter explains in detail about the ontology-based modeling algorithm approach and key owl files construct the algorithm.

4.1 A new approach in algorithm modeling

The mechanism used to model algorithms is based on ontologies. This approach enables the modeling of algorithms decoupling the algorithm itself from the architecture, framework, programming language, and in general from the systems that may host and execute it. To obtain this separation, a set of OWL ontologies has been put in place. The use of ontologies allows modeling the algorithm adding a rich set of semantic information that is not usually included in other modeling techniques and not in programming languages.

In Figure 4-1, there is structure of OWL ontologies.

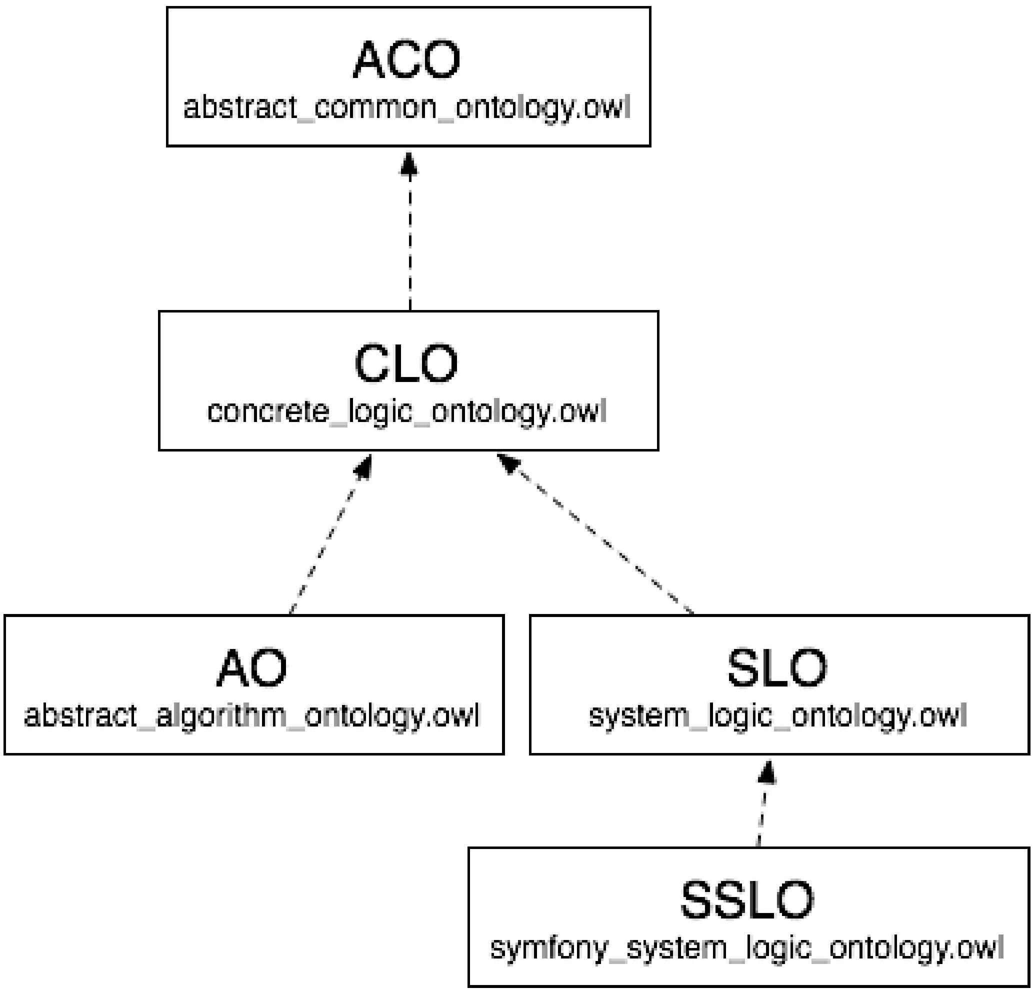

ACO
abstract_common_ontology.owl
CLO
concrete_logic_ontology.owl
AO
abstract_algorithm_ontology.owl
SLO
system_logic_ontology.owl
SSLO
symfony_system_logic_ontology.owl

Figure 4-1: Ontologies-based Algorithm Modelling Architecture

• Abstract Common Ontology: ACO

ACO is needed for technical reason, to store all the common part of CLO. That allows also having the Entity Class available for the CLO.

• Concrete Logic Ontology: CLO

CLO contains the concrete level of the logic ontology, it that includes all the building blocks used to model algorithms. Despite being in the abstract side, it was called concrete to indicate that it contains elements that can be directly mapped to code.

• Abstract Logic Ontology: ALO

ALO is an extension of the CLO and is more abstract. Include elements without a direct translation to code, but that can still be used to create algorithm, adaptations or specify aspects. The abstract structures are mapped to the more concrete elements, part of the CLO, using translation mechanisms.

• System Logic Ontology: SLO

SSLO is the logic ontology OWL structure. This file is needed just to be extended by specific SLO, it that will store the individuals.

Two most important owl file in algorithm modeling

• Algorithm Ontology: AO

AO contains the individuals that constitute an algorithm. Following the OWL suggestion the individuals are stored in a specific ontology file. It extends the

CLO or the ALO, and contains individuals based on them. These individuals constitute a syntax tree acted as a model of an algorithm. So any single AO usually represents an algorithm in the form of a component, and the relative syntax tree. As shown in Figure 4-2 that AO is Tree Structure model contains individual that constitute an algorithm.

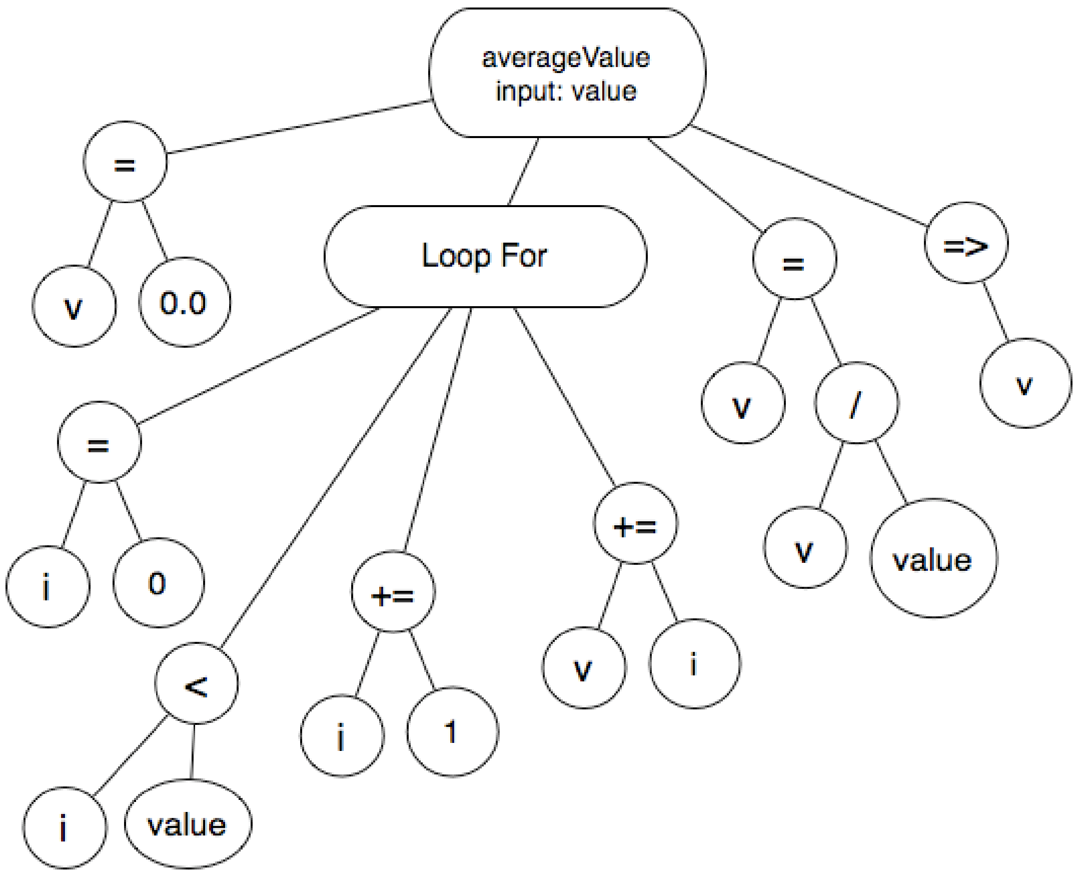
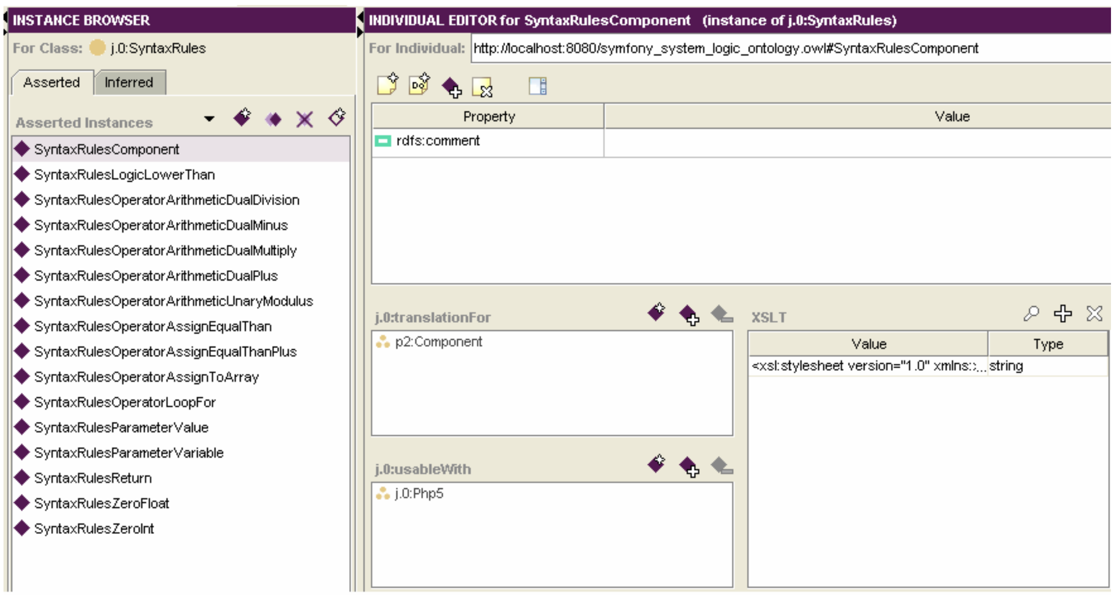


Figure 4-2: AO in Tree Structure

• Specific System Logic Ontology: SSLO

This is specific for different framework/systems. It extends the SLO and includes the individuals that constitute the information needed to map the abstract logic to a specific system. Every element of the CLO, is mapped to code, using XSLT rules, that are stored in the SSLO. An example is Symfony SLO (SSLO), but for other framework a different ontology would be needed (like JSP SLO for a Java JSP framework, or a RSLO for Ruby on Rails). Some deployment rules are added to know how to envelope the code and where to create the proper file to deploy the code for a specific system/framework. SSLO specific for different language, include syntax rules for language feature (control flow, operators, variable, etc) using XSLT as Figure 4-3 shows Protégé snapshot of SSLO.


INSTANCE BROWSER

For Class: j.0:SyntaxRules

Asserted | Inferred

Asserted Instances

- SyntaxRulesComponent
- SyntaxRulesLogicLowerThan
- SyntaxRulesOperatorArithmeticDualDivision
- SyntaxRulesOperatorArithmeticDualMinus
- SyntaxRulesOperatorArithmeticDualMultiply
- SyntaxRulesOperatorArithmeticDualPlus
- SyntaxRulesOperatorArithmeticUnaryModulus
- SyntaxRulesOperatorAssignEqualThan
- SyntaxRulesOperatorAssignEqualThanPlus
- SyntaxRulesOperatorAssignToArray
- SyntaxRulesOperatorLoopFor
- SyntaxRulesParameterValue
- SyntaxRulesParameterVariable
- SyntaxRulesReturn
- SyntaxRulesZeroFloat
- SyntaxRulesZeroInt

INDIVIDUAL EDITOR for SyntaxRulesComponent (instance of j.0:SyntaxRules)

For Individual: http://localhost:8080/symfony_system_logic_ontology.owl#SyntaxRulesComponent

Property	Value
rdfs:comment	

j.0:translationFor

- p2:Component

j.0:usableWith

- j.0:Php5

XSLT

Value	Type
<xsl:stylesheet version="1.0" xmlns:...	string

Figure 4-3: Protégé snapshot of SSLO

Chapter 5 Design

In this chapter, requirements in relation to code generation engine are listed. The introduction of technologies to be used in implementation is also included. XSL style sheets were designed for multiple classes in the logic building block of the algorithm to be used as syntax rule for different language features (loop, if else control flow, variable, function) stored in SSLO. Various functions are designed to handle processes involving in ontology-based code generation and UML diagrams were attached for better explanation.

5.1 Introduction

The purpose of this project was to come up with a solution to generate code (PHP, Java) using a language independent ontology-based abstract algorithm. And code generated can then be deployed on the web server and run as an online service. Main interests and concerns should be associated with the ontology-based algorithm modeling approach to derive the requirements for code generation. In this approach, Algorithm Ontology(AO) is used to represent structure of system and Specific System Logic Ontology(SSLO) used as language specific ontology. For final generated code, a convenient way for deployment should be considered. Decisions also need to be made in choosing technologies that are most suitable for implementation this project.

5.2 Requirements

5.2.1 Requirement for SSLO

As we already mentioned in the last chapter, SSLO is specific for each language, include syntax rules for language features (control flow, operators, variable, etc) using XSL. Since SSLO structure is already defined, the missing part was to design and implement XSL for each syntax rules used in Java and PHP SSLO.

5.2.2 Requirements for Code Generation Engine

1) Requirement for AO tree traversing

To have a way for AO tree traversing is very important since the AO tree nodes can be treated in four modes as Root Element, Current Element, Sub Element, and Next Element. First of all, the requirement for AO tree parsing is to find the Root Element of AO tree as a starting point. Secondly, make root Element to be current element. Thirdly, condition should be designed to check whether the current Element in the tree has Sub Element or Next Element linked. Finally, CGE should be able to know when to finish code generation which also means find the last element in the AO tree that it doesn’t has either Sub Element(s) or Next Element.

2) Requirement for retrieving XSL from SSLO

Next stage is to transform each element to code. In AO, each element represents as an individual of a certain class. Those classes are designated to represent language features (control flow, operator, variable, etc). Requirement here is that each individual in AO should be associated with a syntax rule in the SSLO. Finding the matched syntax rule in SSLO will derive XSL for specific individual transformation.

3) Requirement for create a buffer to build template code and continuously added up to final output

Because AO tree is constructed by set of elements, those elements are transformed one by one from root to the end. So a buffer is required to continuously add generated code to it. After the last element is transformed, buffer can then become the final output.

4) Requirement for generated code deployment

In order to test whether generated code especially PHP code is able to work on the web server. Code deployment should be concerned that it should wrap the generated code with some necessary information to make a complete PHP file and deploy the file to a specific server folder so that it can then be access through the web browser.

5) Requirement for designing system with user friendly interface
A user friendly interface is necessary for CGE that it should allow user to select the AO, choose output language type, and also make it easy for user to deploy the generated code to server.

6) Requirement for providing choice of PHP or Java code generation
Because CGE is required to support PHP and Java code generation, users should have the right to choose kind of output language as they want. The point need to be concerned to make sure that CGE is able to generate code for PHP and Java based on the same AO.

7) Requirement for supporting dynamic and pluggable web service
Supporting the dynamic and pluggable web service is one of research aims. Because of time constraint that we can not support very sophisticated web service like SOAP, WSDL, UDDI which they are widely used in the industry but at least we can show that CGE is able to support some level of dynamic and flexible that changes made to AO will lead to change to the online service (based on regenrated code from modified AO).

5.3 Techniques used for CGE

From research on the internet which meaning the in the state of Art, I found the Protégé API and Saxon API can be very much suitable to be project implementation and this section is to explain some details about these two APIs and what functionality they can provide.

5.3.1 Protégé API

The Protégé-OWL API is an open-source Java library for the Web Ontology Language

(OWL) and RDF(S). The API provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning based on Description Logic engines. Furthermore, the API is optimized for the implementation of graphical user interfaces [17].

Protégé-OWL API is a set of Java interfaces from the model package. Those interfaces provide access to the OWL model and its elements like classes, properties, and individuals [17].

Among all the interfaces in the Protégé package, the most important model interface is OWLModel, which it provides access to the top-level container of the resources in the ontology. OWLModel also can be used to create, query, and delete resources of various types and then use the objects returned by the OWLModel to do specific operations.
In Protégé API, Named classes are used to create individuals, sample code for create a OWLNamedClass is

OWLNamedClass personClass=owlModel.createOWLNamedClass("Person");

OWLModel can also be used to query and traverse the contents of an ontology shown in Figure 5-1. The following code detail explain how query is achieved through iteration of all OWLClasses of an ontology.

```
    String uri = "http://www.owl-ontologies.com/travel.owl";
    OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
    Collection classes = owlModel.getUserDefinedOWLNamedClasses();
    for (Iterator it = classes.iterator(); it.hasNext();) {
        OWLNamedClass cls = (OWLNamedClass) it.next();
        Collection instances = cls.getInstances(false);
              for (Iterator jt = instances.iterator(); jt.hasNext();) {
            OWLIndividual individual = (OWLIndividual) jt.next();
        }
    }
```
Figure 5-1: Code Sample of traversing an ontology using Protege API

5.3.2 Saxon and Saxon API

Saxon is Open Source XSLT processor which developed by Michael Key[18]. It is used to translate the XML style document using XSLT stylesheet. The version I chose for implementing this Code Generation project is Saxon 6.6.5 as it supports XSLT 1.0 and XPath.
The using of Saxon is very straightforward that once you downloaded Saxon from website and then unzip the binary version to get a Jar file named as saxon.jar.

There are two ways to run Saxon based on the command line
1) java –jar saxon.jar source.xml stylesheet.xsl
But –jar option which make classpath ignored

2) If saxon.jar is included in the classpath, then just run Saxon using the command
java com.icl.saxon.StyleSheet source.xml styesheet.xsl
source.xml is xml file which waited to be translated
stylesheet.xsl is the XSL stylesheet describe how to translate source.xml

The output can be any kind files. One of most common used output type is html, but it really depends on what stylesheet.xsl describes to determine the type of output[18].

Saxon also includes a Java library which it supports a similar processing model to XSL, so that Java developers can use Saxon API as third part tool to build their own project[18]. One of major advantages by using the Saxon API is because it includes DOM, SAX and JAXP as standard to enable parse, transform, validate and query XML documents which provide rich function sets to developer to use and continue development. In this project, Saxon XSLT Processor is used to interpret AO files with XSLT style sheet retrieved from the syntax rule part of the SSLO to generate code the

Saxon XSLT Process as shown in Figure 5-2

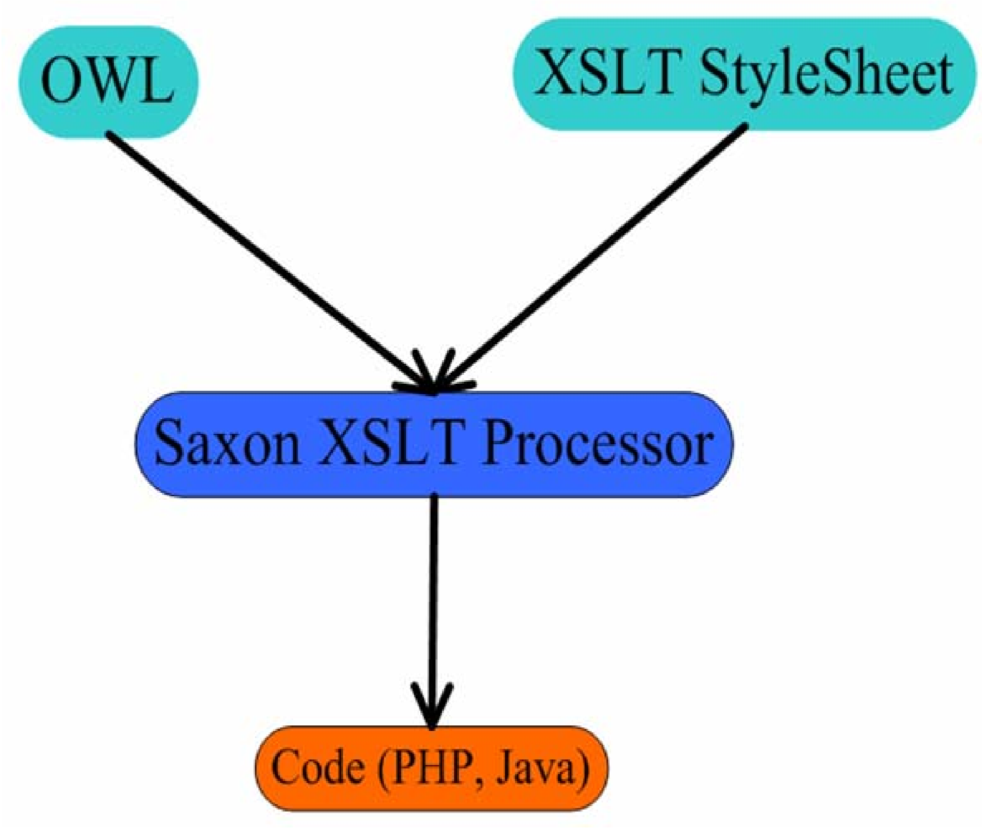


Figure 5-2: Saxon XSLT Processor used in CGE

5.4 XSL designed for classes in the logic building block of the algorithm

XSL style sheets were designed for multiple classes in the logic building block of the algorithm to be used as syntax rule for different language features (loop, if else control flow, variable, function) stored in SSLO.

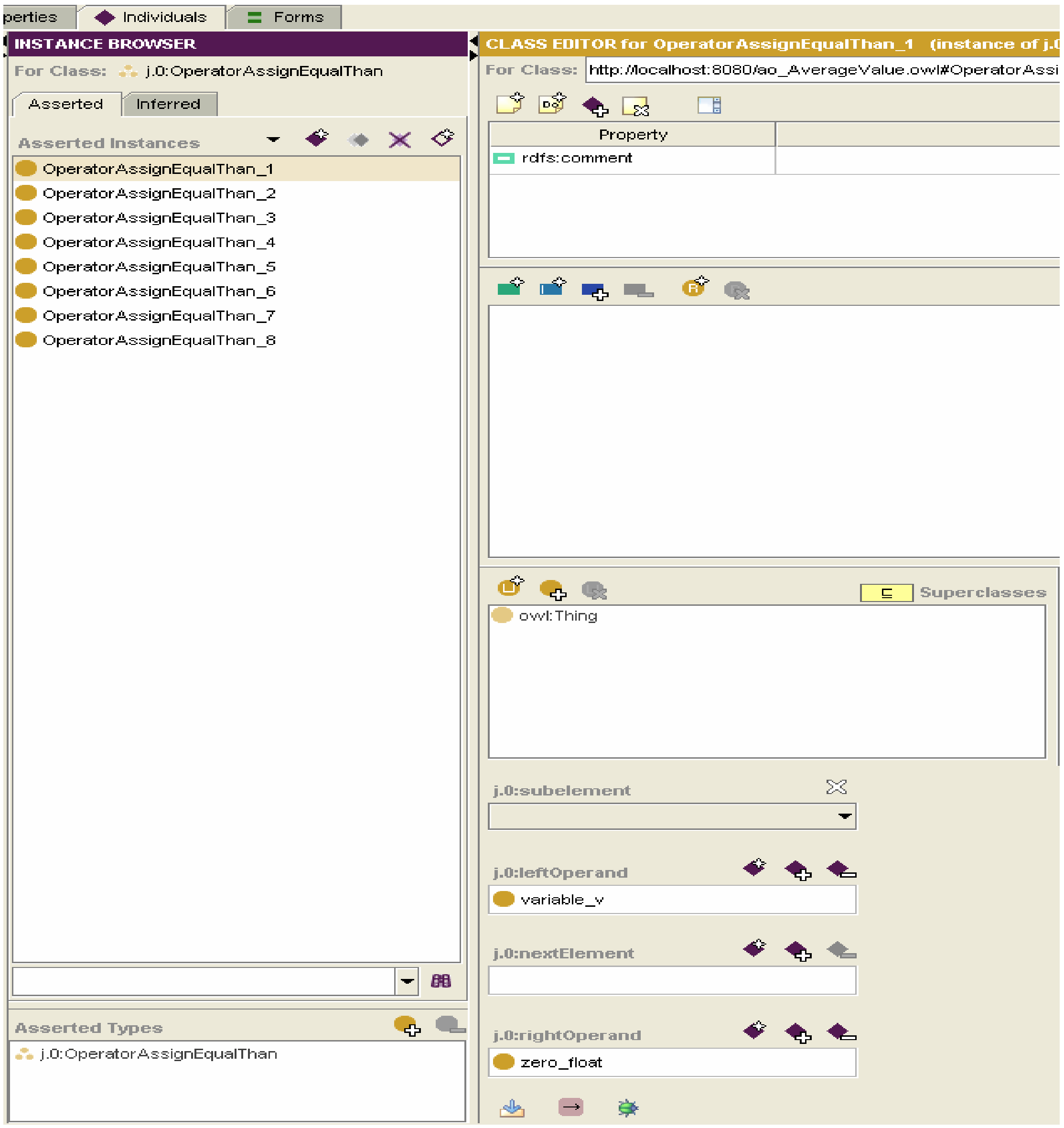
5.4.1 Design XSL for Class OperatorAssignX

Class OperatorAssignEqualThan, OperatorAssignEqualThanPlus, OperatorAssignToArray, OperatorAssignFullArray are defined in the logic building block of the algorithm Equal operator (A = B) or EqualThanPlus operator (A =+ B), they share similar structure that they are subclasses of Class OperatorAssign. As we can see from following Protégé snapshot (Figure 5-3) that they both have same set of properties.

XSL stylesheet to transform instance (individual) of these classes should match “j.0: leftOperand” and “j.0: rightOperand” properties in the AO to retrieve values associated with these two properties to get code generated as pattern like

{@leftOperand@} = {@rightOperand@}

The only difference for those class here is from change “=” to “=+” for OperatorAssingEqualThanPlus. Property “j.0:subelement” value is true or false, and determinate whether to add “;” as closing symbol as part of the generated code.



perties
Individuals
Forms
INSTANCE BROWSER
For Class: j.0:OperatorAssignEqualThan
Asserted
Inferred
Asserted Instances
OperatorAssignEqualThan_1
OperatorAssignEqualThan_2
OperatorAssignEqualThan_3
OperatorAssignEqualThan_4
OperatorAssignEqualThan_5
OperatorAssignEqualThan_6
OperatorAssignEqualThan_7
OperatorAssignEqualThan_8
Asserted Types
j.0:OperatorAssignEqualThan
CLASS EDITOR for OperatorAssignEqualThan_1 (instance of j.0
For Class: http://localhost:8080/ao_AverageValue.owl#OperatorAssi
Property
rdfs:comment
Superclasses
owl:Thing
j.0:subelement
j.0:leftOperand
variable_v
j.0:nextElement
j.0:rightOperand
zero_float

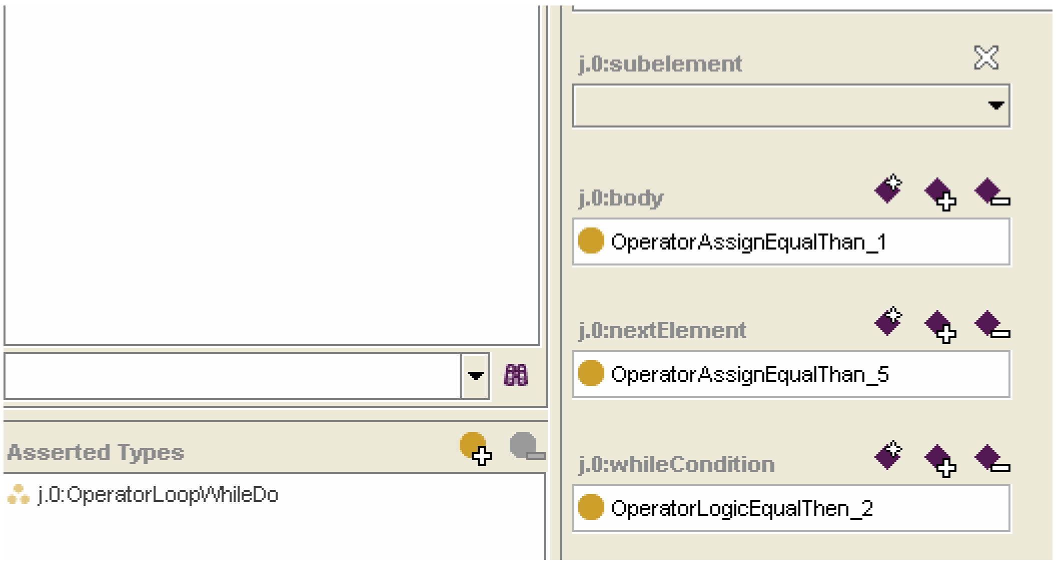
Figure 5-3: Protege SnapShot of Class OperatorAssignEqualThan

5.4.2 Design XSL for Class OperatorLoopDowhile or

OperatorLoopWhileDo

Class OperatorLoopDowhile and Class OperatorLoopWhileDo defined in the logic building block of the algorithm to model "do while" or "while do" loop. As we can seen from the following Protégé snapshot (Figure 5-4) that any individual of Class OperatorLoopDoWhile or OperatorLoopWhileDo share same set of properties, the two most important properties are "j.0:body" and "j.0:whileCondition". XSL for both classes should match these two properties under one specific OperatorLoopDowhile individual (passed-into XSL as a parameter to tell which OperatorLoopDowhile individual is waiting to be transformed). The final generated code contains a pattern like

do{@Body@} while({@condition@})	is for Class OperatorLoopDoWhile
while{@condition@} do {@body@}	is for Class OperatorLoopWhileDo



Asserted Types
j.0:OperatorLoopWhileDo

j.0:subelement

j.0:body
OperatorAssignEqualThan_1

j.0:nextElement
OperatorAssignEqualThan_5

j.0:whileCondition
OperatorLogicEqualThen_2

Figure 5-4: Protege SnapShot of Class OperatorLoopWhileDo

5.4.3 Design XSL for Class OperatorLoopFor

Class OperatorLoopFor is defined in the logic building block of the algorithm to model for loop. As we can seen from the following Protégé snapshot (Figure 5-5) that any individual of Class OperatorLoopFor has six properties, the four most important properties are “j.0:startingElement”, “j.0:conditionElement”, “j.0:incrementStateElement” and “j.0:body”. XSL for this class should match these four properties under one OperatorLoopFor individual (passed-into XSL as a parameter to tell which OperatorLoopFor individual is waiting to be transformed). The final generated code should has pattern like

for({@startingElement@}, {@conditionElement@},

{@incrementalElement@}){ {@body@ }”

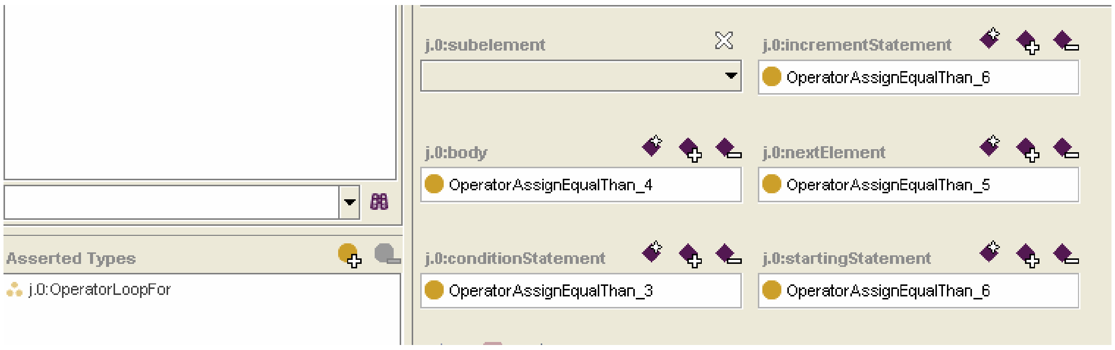


Figure 5-5: Protege SnapShot of Class OperatorLoopFor

5.4.4 Design XSL for Class OperatorCondtionDualIfThen and Class OperatorCondtionDualIfThenElse

Class OperatorCondtionDualIfThen and OperatorCondtionDualIfThenElse are defined in the logic building block of the algorithm to model if else control flow. As we can seen from the following Protégé snapshot (Figure 5-6) that any individual of Class OperatorCondtionDualIfThen has four properties and two most important properties are “j.0:condtion” and “j.0:fristBody”. XSL for classs OperatorCondtionDualIfThen should match these two properties under one specific OperatorCondtionDualIfThen individual (passed-into XSL as a parameter to tell which OperatorCondtionDualIfThen individual is waiting to be transformed). The final generated code should has pattern like

if({@condition@}) { {@firstBody@} }

Class OperatorCondtionDualIfThenElse has extra property “j.0:secondBody” so that it support generated code in pattern like

if({@condition@}) { {@firstBody@} else {@secondBody@}}

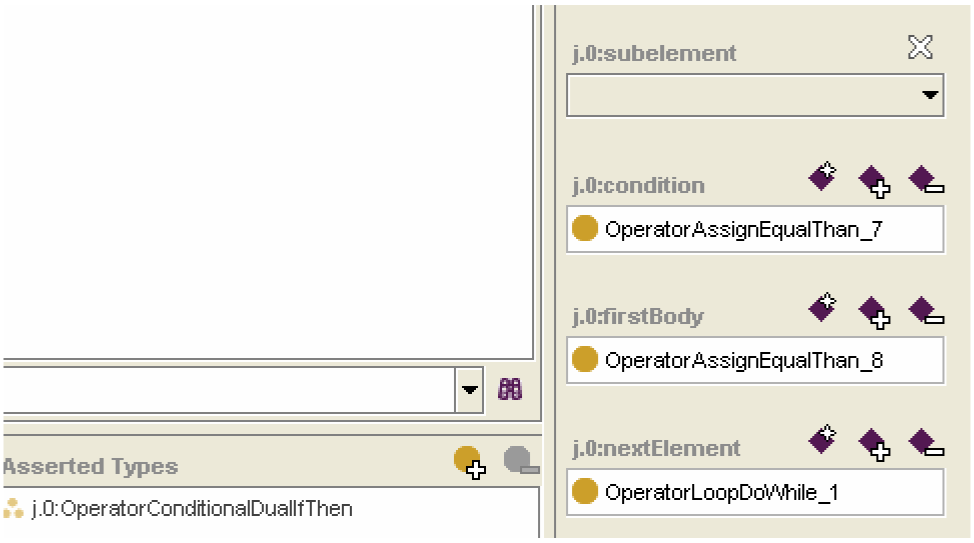


Figure 5-6: Protege SnapShot of Class OperatorCondtionalDualIfThen

5.4.5 Design XSL for Class Component

Class Component is defined in the logic building block of the algorithm to model function. As we can seen from the following Protégé snapshot (Figure 5-7) that any individual of Class Component has seven properties and the two most important properties for Class Component transformation to PHP code are “j.0:hasParameter” and “j.0:componentBody”. XSL for Class Component should match these two properties under one specific Component individual (passed-into XSL as a parameter to tell which Component individual is waiting to be transformed). The final generated code should has pattern like

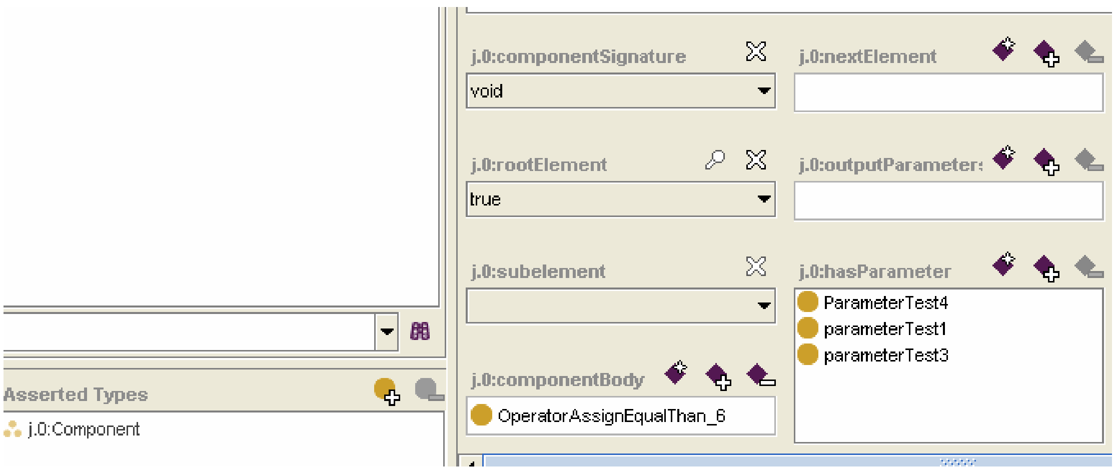
function functionname ({@parameter1@},{@parameter2@})

{ {@componentBody@} }

For XSL of Class Component designed for Java code that property “j.0:componentSignature” is also included that because each Java function needs to specify signature and return type which are quite different from PHP. Code generated for Java looks like pattern

public void function functionname ({@parameter1@},{@parameter2@})

{ {@componentBody@} }



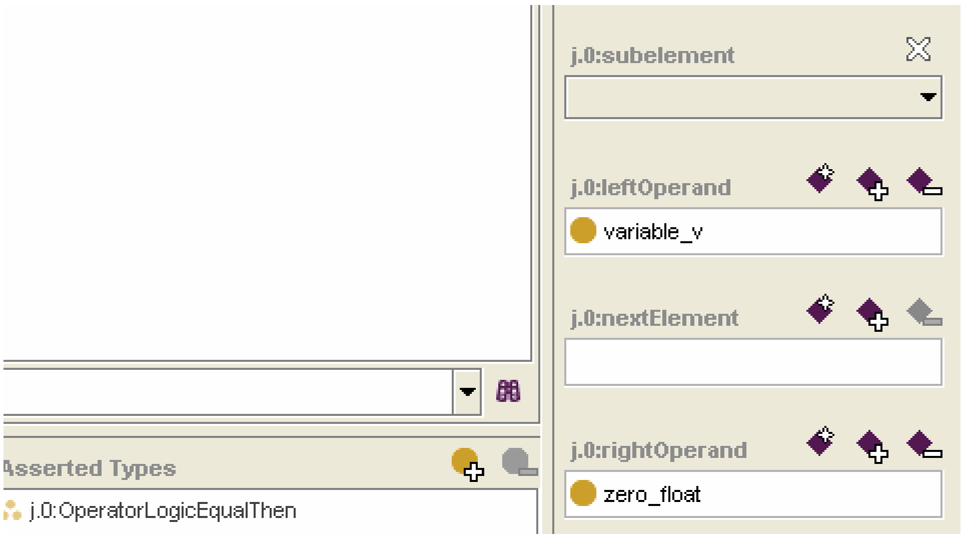
j.0:componentSignature: void
j.0:rootElement: true
j.0:hasParameter: ParameterTest4, parameterTest1, parameterTest3
j.0:componentBody: OperatorAssignEqualThan_6

Figure 5-7: Protege SnapShot of Class Component

5.4.6 Design XSL for Class OperatorLogicX

Class OperatorLogicEqualThen, OperatorLogicGreaterEqualThen, OperatorLogicGreaterThen, OperatorLogicLowerEqualThen, OperatorLogicLowerThen, OperatorLogicNotEqualThen are defined in the logic building block of the algorithm to model logic operator such as A>B, A<B, A<=B, etc. They share similar structure that they are subclasses of Class OperatorLogic. As we can see from following screen cut from Protégé (Figure 5-8) that they both have same set of properties. XSL style sheet to transform instance (individual) of these classes should match “j.0: leftOperand” and “j.0: rightOperand” properties in the AO.owl under one specific OperatorLogicX individual (passed-into XSL as a parameter to tell which OperatorLogicX individual is waiting to be transformed). The final code generated has pattern as

{@leftOperand@} > {@rightOperand@}



variable_v

zero_float

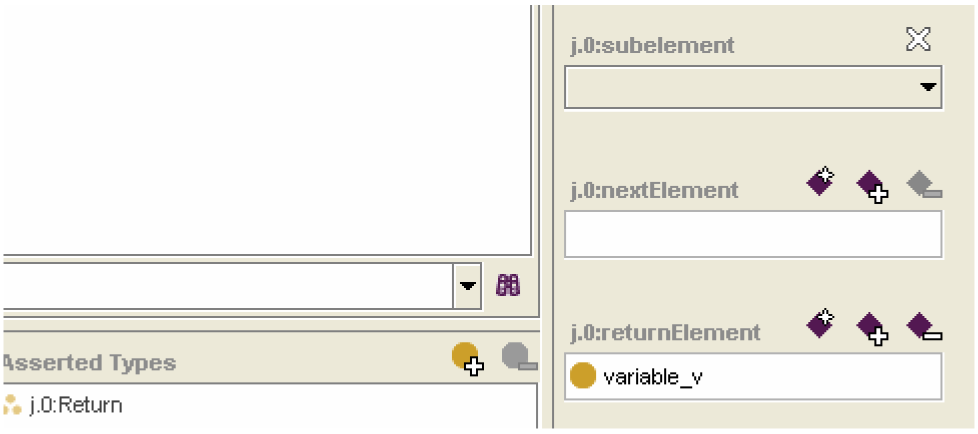
j.0:OperatorLogicEqualThen

Figure 5-8: Protege SnapShot of Class OperatorLogicEqualThen

5.4.7 Design XSL for Class Return

Class Return is defined in the logic building block of the algorithm to model return of a function. As we can seen from the following Protégé short cut(Figure 5-9) that any individual of Class Return has three properties and the most important property for Class Return transformation is "j.0:returnElement". XSL for Class Return should match this property under one specific Return individual (passed-into XSL as a parameter to tell which Return individual is waiting to be transformed). The final generated code should have pattern like

return {@parameter1@};



variable_v

j.0:Return

Figure 5-9: Protege SnapShot of Class Return

5.4.8 Design XSL for Class Variable

Class Variable is defined in the logic building block of the algorithm to model a variable. As we can seen from the following Protégé short cut (Figure 5-10) that any individual of Class Parameter has four properties and the most important property for Class Parameter transformation is "j.0:datatype". "j.0:datatype" tells what variable type is. XSL for Class Parameter should match this property under one specific Parameter individual (passed-into XSL as a parameter to tell which Parameter individual is waiting to be transformed). The final generated code should have pattern like

${@parameter1@}	for PHP
{@parameter1@}	for Java

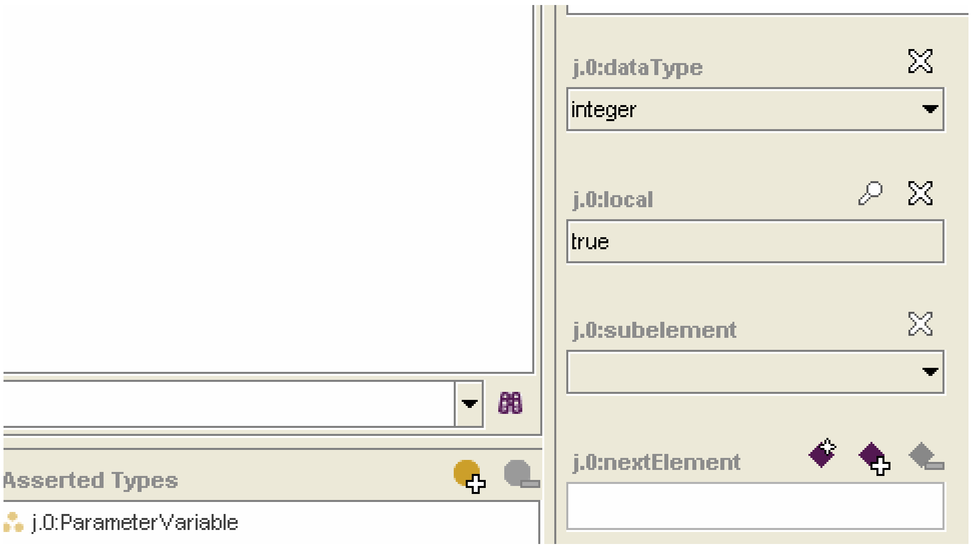

j.0:dataType
integer
j.0:local
true
j.0:subelement
j.0:nextElement
Asserted Types
j.0:ParameterVariable

Figure 5-10: Protege SnapShot of Class ParameterVariable

5.5 Design for Code Generation Engine

5.5.1 Code Generation Architecture

As we can see from the Figure 5-11 which shows the overall architecture for code generation, this architecture can be mainly divided into three parts. In the diagram, we can see the left two components are AO and SSLO owl as initial input for CGE to take. In the middle of diagram, it is the CGE itself that it interprets AO using SSLO and generate source code (Java or PHP). On the right side of diagram includes the Deployment Batch Procedure that it is able to build actual PHP or Java files based on generated code. And those PHP or Java files can then be output to the web server folder to run as an online service. This procedure well decouples AO from final output code and make CGE language independent.

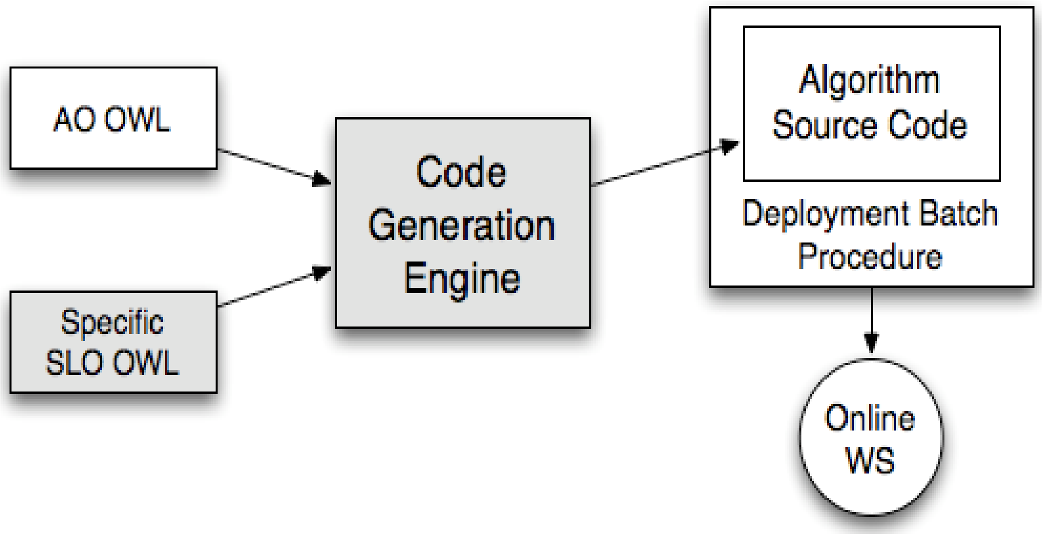


Figure 5-11, Code Generation Architecture

5.5.2 Deeper understanding of AO tree

Before entering the detail design stage, it is necessary to review the AO tree to get deeper understanding of it. The reason for that is to make clear the definition of RootElement, CurrentElement, SubElement(s), NextElement in the AO.

The following diagram (Figure 5-12) visually showing the tree structure of AO, individual component_averageValue is first Element in this tree which it is assigned to be Root Element. Once we found the RootElement in the tree then we assign it to be Current Element for transformation. From the RootElement, first left element links to it is OperatorAssignEqualThan_6 can be seen as a SubElement of component_averageValue. And in the diagram, we can see that OperatorAssignEqualThan_6 has two leave nodes as variable_v and zero acting as its SubElements. From Operator AssignEqualThan_6, there is arrow point to individual

OperatorLoopFor_8 with name nextElement which it means invidual OperatorLoopFor_8 is the nextElement of invididual OperatorAssignEqualThan_6.

Value of property ‘nextelement’ of OperatorAssignEqualThan_6 is now set to be OperatorLoopFor_8 making OperatorLoopFor_8 becomes OperatorAssignEqualThan_6’s next elment. By filling nextElement property, it will automatically assign an individual to be next element of current invididual in the AO. At the end of tree which is on very righthand side of diagram which is individual variable_v. variable_v is the SubElement of individual Return_7. variable_v doesn’t have any NextElement or SubElement so that once it was transformed then we can confirm reaching the ending the code generation for a specific AO. From Figure 5-12 we can conclude that each Element in the AO tree can either have zero to multiple SubElement(s) but can only have zero to one NextElement and itself can be either SubElement or NextElement of Element ahead of it except RootElement.

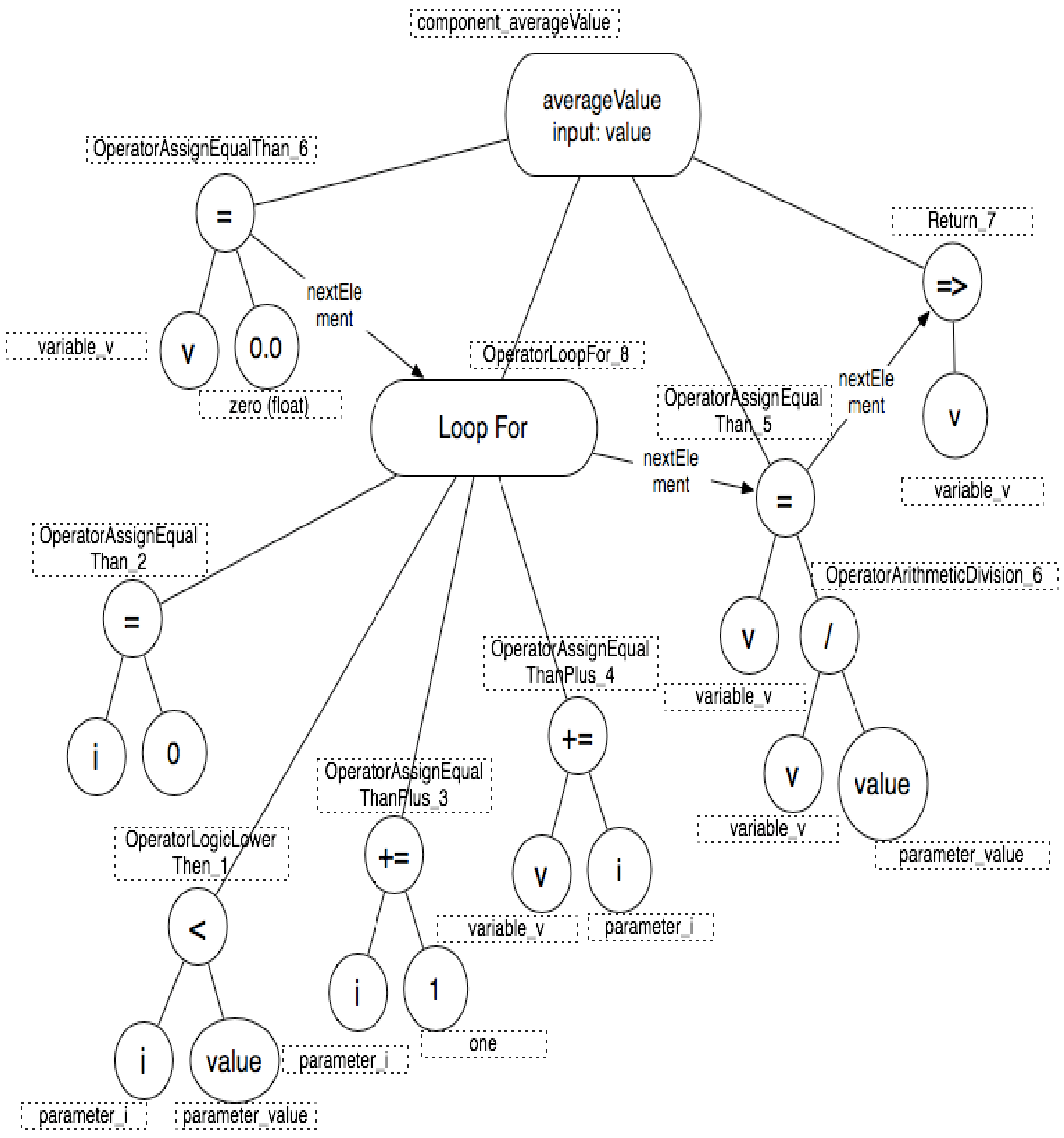
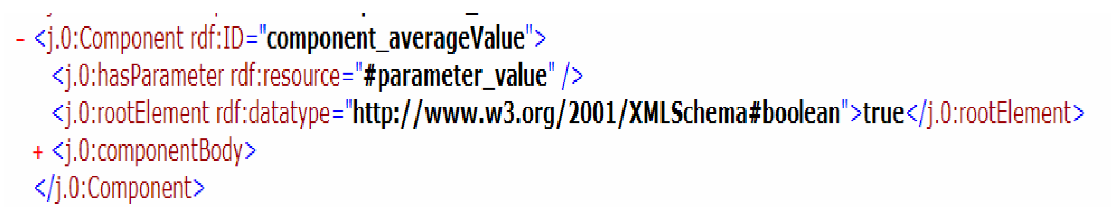


Figure 5-12, AO Tree Structure in detail

5.5.3 Code Generation Engine Functional Design

1) Function Design for finding the RootElemnt

To find the Root Element in the AO tree is starting point of code generation processes. Since Protégé API supports iterating all the individuals in the AO one by another, Function designed should include condition check to find the Root Element. If any individual property j.0:rootelement value is true so that this individual is then assigned to be Root Element of the whole AO tree.



```
- <j.0:Component rdf:ID="component_averageValue">
    <j.0:hasParameter rdf:resource="#parameter_value" />
    <j.0:rootElement rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</j.0:rootElement>
  + <j.0:componentBody>
  </j.0:Component>
```

Figure 5-13, Sample AO contains rootElement

2) Function for Current Element transformation

Function designed for Current Element transformation should include four processes as depicted in Figure 5-14. The first is to find matched individual of Class SyntaxRule for Current Element in SSLO, the matching is based on ‘transformFor’ property of each SyntaxRule individual. If any individual’s ‘transformFor’ property is equal to Current Element then the individual matches. The second process it to retrieve the XSLT from found individual getting the value of its ‘XSLT’ property. The third step is to send XSLT and AO to the Saxon XSLT processor to transform the current element. The final process is to buffer the return string from Saxon for further transformation.

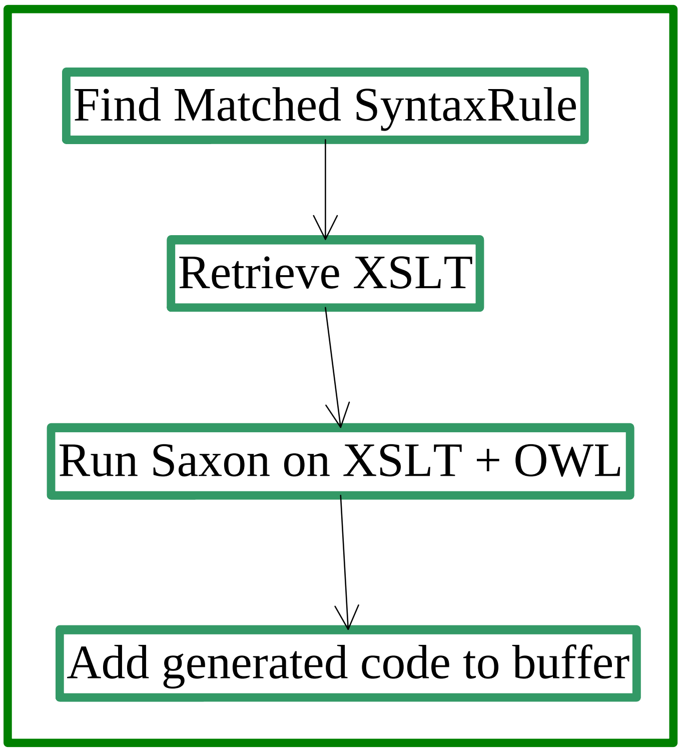


Figure 5-14, Current Element Transformation Processes

3) Function for Pattern Search

As we already discussed in the previously that all of elements in the AO are transformed one by another following the AO tree structure, Current Element can have zero to multiple SubElement(s) and zero or one NextElement if this element is not the last element of tree. So now it left problem that how to judge Current Element has SubElement or NextElement. For NextElement, the solution is very straight forward, because if each current Element (individual) has property "NextElement" and if property value is not null which means there is another Element linked to Current Element. However when it turns to judge whether current Element has SubElement(s) or not, it becomes a little bit complicate. A new way of introducing special pattern {@ @} to wrap the SubElements that are not transformed yet

We can take individual OperatorAssignEqualThan_6 as an example that variable_v and zero are two SubElements of OpertorAssignEqualThan_6. variable_v is described as leftOperand, zero is described as rightOperand of OperatorAssignEqualThan_6, the XSL for Class OperatorAssignEqualThan should retrieve value of the leftOperand and rightOperand properties. After OperatorAssignEqualThan_6 is transformed, {@variable_v@} = {@zero@} is return and becomes buffer string, specific pattern "{@ @}" is added to both leftOperand and rightOperand property value to represent variable_v and zero are SubElements of OperatorAssignEqualThan_6. We can also say that any individual which wrapped by pattern {@ @} , means it needs to be transformed.

At this point, Regular Expression is introduction to provide solution for pattern search that class called StringRegualr is designate to handle this task that it has parseOrignialString(buffer String) method to detect whether {@ @} pattern exist in the passed-in parameter buffer String. If pattern exists, then it will retrieved the name of first individual wrapped by the special pattern and return it. Otherwise it will return string "patternNotFound" noticing that is buffer String is fully transformed.

4) Functions for Recursive transformation

Because the AO individuals forms a tree structure, it is very important to have recursive functions for travesing that tree and call the transformation on each element. The recursive function keeps on calling itself until certain circumstances (no subelement and no next element) are reached. I designed two recursive functions for the CGE. One recursive function is to call transformation method on Current Element and call second recursive function to travser all the sublement of current element and call transformaton method on each of subelement. Once all the subelements of Current Element are transformed, then Current Element check whether it has NextElement or not. If Current Element has NextElement, Current Element calls the recursive first function on the NextElement, so NextElement becomes new Current Element. if Current Element transformed all its SubElements and there is no NextElement linked to it, the process stop. Then it is confirmed that we are at the end of the AO tree, and the buffer contains the final code output. The following state diagram Figure 5-15 clearly explains the recursive transformation processes flow.

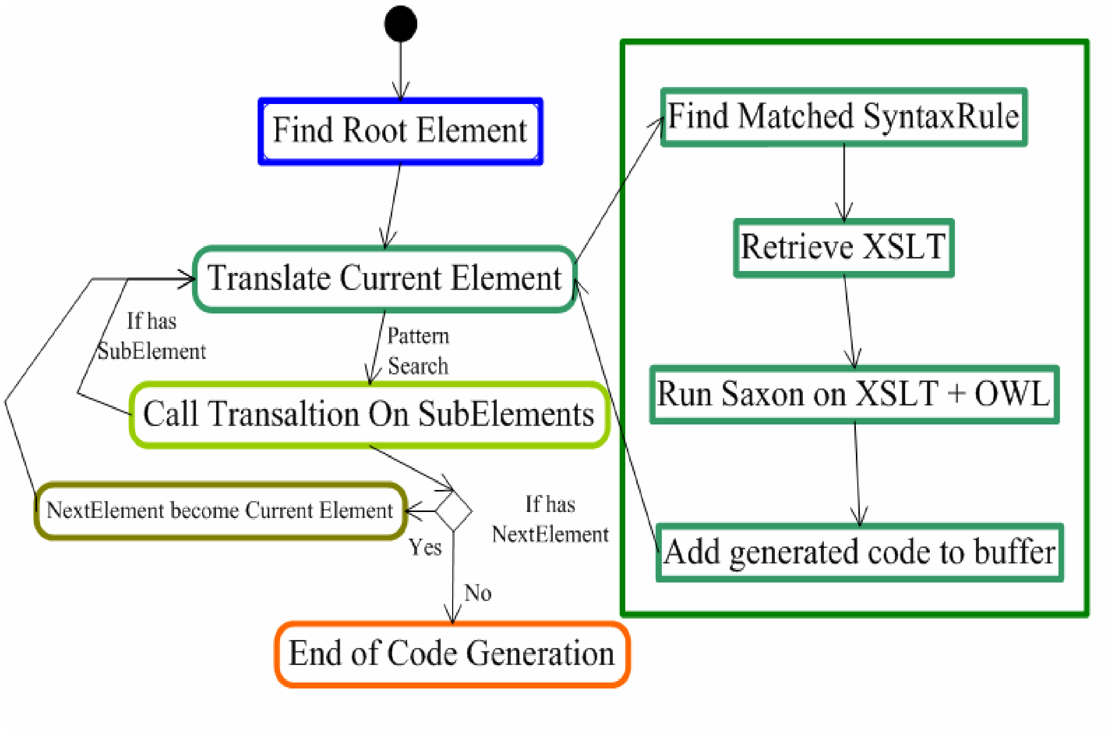


Figure 5-15: Recursive Transformation Processes Flow

5) Function for code deployment

Once we finished code generation, then next step is to make generated cod to be become a runnable file and deploy it to server folder. So function for code deployment is majorly doing two jobs. First jobs is make code generated (string type) through CGE to be an PHP or Java file. One thing should be concerned here is that PHP file normally starting with ‘<?php’ and ending with ‘?>’. And Java file is starting with ‘public class ClassName {‘ and ending with ‘}’. At the moment, CGE is only able to generated code based on what AO describes. The AO doesn’t include any information about ‘<?php’ or ‘?>’, so that the code generated from CGE is not sufficient to be run directly on the server and need to be wrapped with “<?php” and “ ?>” to build a complete PHP file. And AO filename is derived and becomes the php file name.
For Java, I designed and made AO filename to be ClassName. Then wrap “public class ClassName {“ and “}” with generated code to build a complete Java file.

Second point of function is to design a function to deploy php file to server folder, A FileOutputStream is necessary to be created to deploy php file to a specific server folder. Then it can then be access from internet browser.

6) User interface Design

A user friendly graphic interface is also concerned very important as a part of CGE for rich user experience. Java Swing is chose to be implementation technologies, and GUI are created to handle basically two kind of major work. Firstly, two menu items created to allow user to choose either PHP to Java to be final output language. Then FileChooser is designed to pop out for user to choose the AO.owl from the file system. Final generated code is shown in the textarea in the GUI shown in Figure 5-16. And if user want to deploy the generated code, another menu item is designated to handle this task that once user click this “Make File” menu item, the generated code will be automatically deployed to server folder.

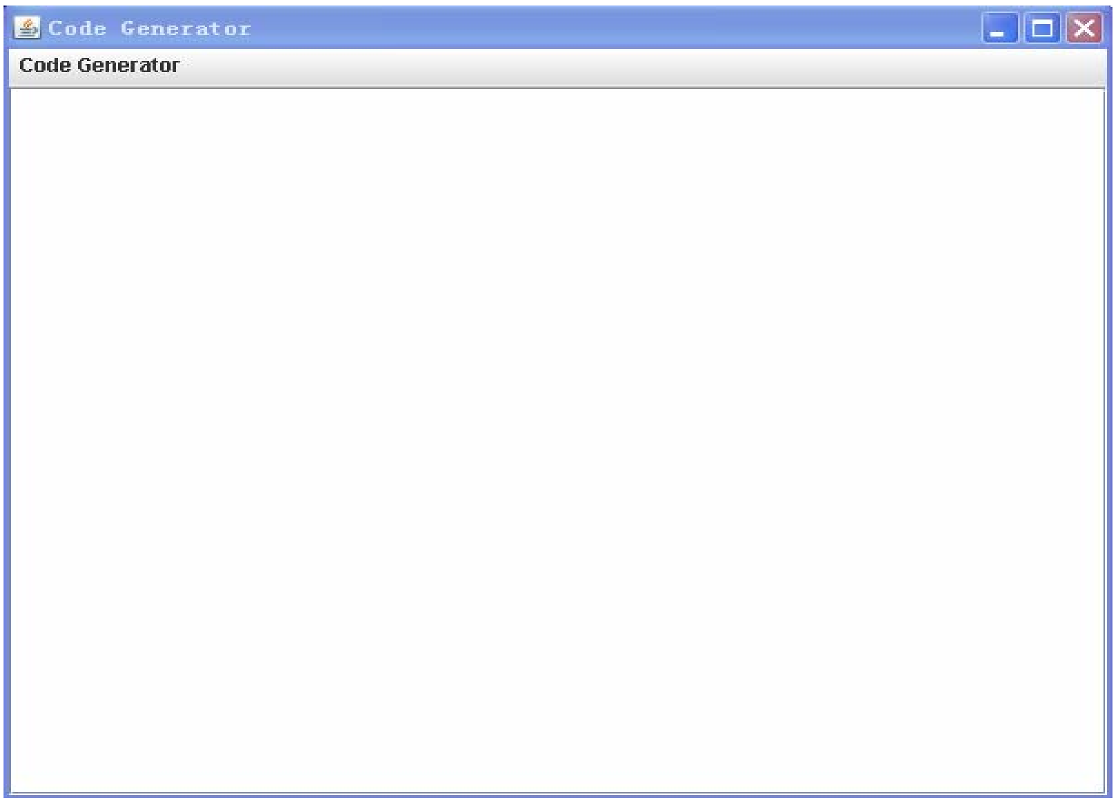


Figure 5-16: GUI of CGE

Chapter 6 Implementation

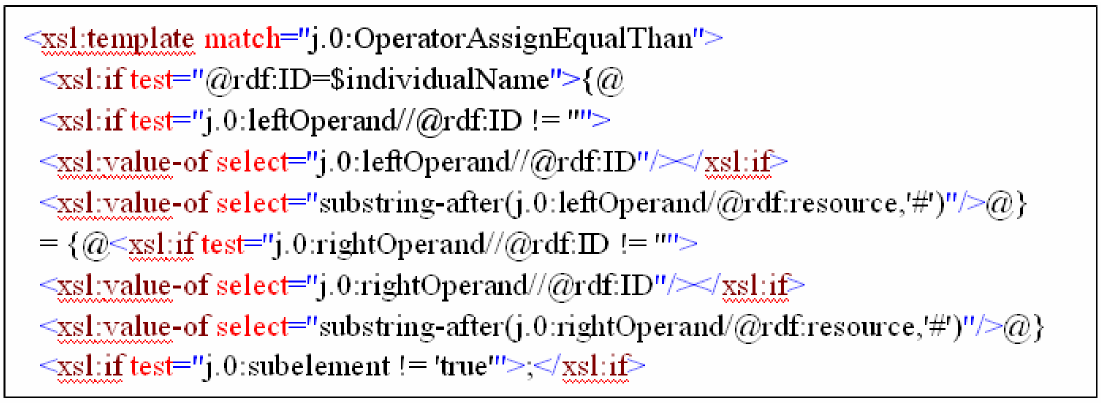
This chapter firstly explains implementation of SSLO for PHP and Java language features. And then it is concentration on explanation in detail of how CGE was implemented. Various classes and functions were development for different purposes and explanation they are interacted to make CGE work.

6.1 SSLO Implementation

The section is to explain in detail how SSLO is implemented and explain some of the used XSL. The classes in the Ontology-based Algorithm need to be associated to a XSLT rule for each specific lanauge feature (control flow, variable, operator, etc…) in order to be translated to proper code. The programming of XSL for each language feature is a common work between Simone and me that to write the XSLs and test them with different AO to make sure that these XSLs are able to transform all individuals in an AO to be either Java or PHP code. As already mentioned in design chapter that these XSLs will be explained one by another in details.

6.1.1 XSL implementation for Class OperatorAssignX

Class OperatorAssignEqualThan, OperatorAssignEqualThanPlus, OperatorAssignToArray, OperatorAssignFullArray are defined in ontological algorithm to model equal operator (A = B) or equal than plus operator (A =+ B).



```
<xsl:template match="j.0:OperatorAssignEqualThan">
 <xsl:if test="@rdf:ID=$individualName">{@
 <xsl:if test="j.0:leftOperand//@rdf:ID != ''">
 <xsl:value-of select="j.0:leftOperand//@rdf:ID"/></xsl:if>
 <xsl:value-of select="substring-after(j.0:leftOperand/@rdf:resource,'#')"/>@}
 = {@<xsl:if test="j.0:rightOperand//@rdf:ID != ''">
 <xsl:value-of select="j.0:rightOperand//@rdf:ID"/></xsl:if>
 <xsl:value-of select="substring-after(j.0:rightOperand/@rdf:resource,'#')"/>@}
 <xsl:if test="j.0:subelement != 'true'">;</xsl:if>
```

Figure 6-1: XSL implentation for Class OperatorAssignEqualThan

As we can seen from the above XSL piece of code shown in Figure 6-1 that the XSL template is created to search for all individuals of type "j.0:OperatorAssignEqualThan". And individualName is parameter passed as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. "j.0:leftOperand" and "j.0:rightOperand" properties are then matched under this specific individual to derive the values of these two properties. Finally generated code for this specific OperatorAssignEqualThan individual looks like:

{@leftOperand@} = {@rightOperand@}

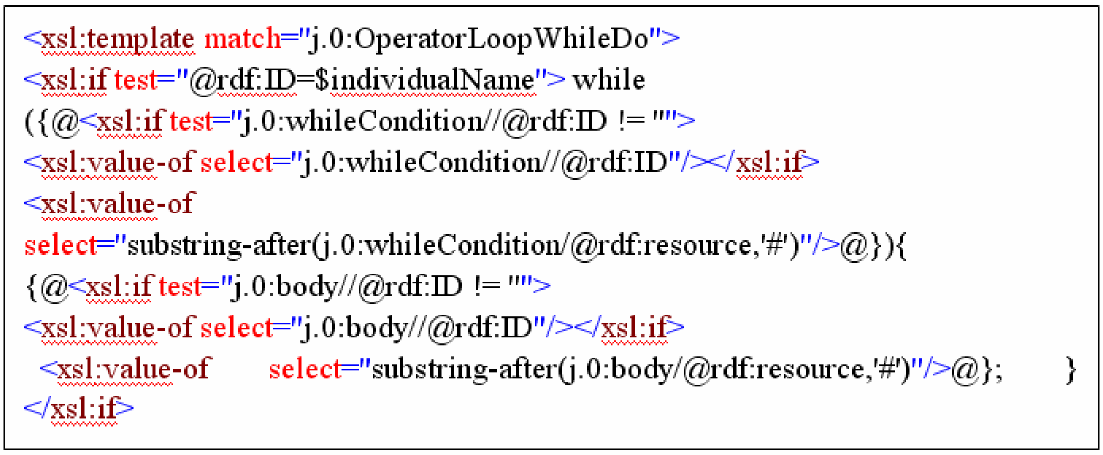
One thing has to be mentioned, there is a test of the property "j.0:subelement" in the last line, the symbol ";" is added to the end of generated code if value is not true So generated code looks like

{@leftOperand@} = {@rightOperand@};

Otherwise, if "j.0:subelement" value is true, there is no semicolon at the end.

6.1.2 XSL implementation for Class OperatorLoopDowhile or OperatorLoopWhileDo

Class OperatorLoopDowhile, OperatorLoopWhileDo, are defined in ontological algorithm to model 'do while' or 'while do' loop.



```
<xsl:template match="j.0:OperatorLoopWhileDo">
<xsl:if test="@rdf:ID=$individualName"> while
({@<xsl:if test="j.0:whileCondition//@rdf:ID != ''">
<xsl:value-of select="j.0:whileCondition//@rdf:ID"/></xsl:if>
<xsl:value-of
select="substring-after(j.0:whileCondition/@rdf:resource,'#')"/>@}){
{@<xsl:if test="j.0:body//@rdf:ID != ''">
<xsl:value-of select="j.0:body//@rdf:ID"/></xsl:if>
  <xsl:value-of      select="substring-after(j.0:body/@rdf:resource,'#')"/>@};      }
</xsl:if>
```

Figure 6-2: XSL implementaiton for Class OperatorLoopWhileDo

As we can see from the above XSL piece of code shown in Figure 6-2 that the XSL template is created to search for all individuals of type "j.0: OperatorLoopWhileDo". And individualName is parameter passed as an input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. Value of Property "j.0:whileCondition" is then derived under this specific individual. And value of property "j.0:body" is retrieved as well. Finally generated PHP code for this specific OperatorLoopWhileDo individual looks like

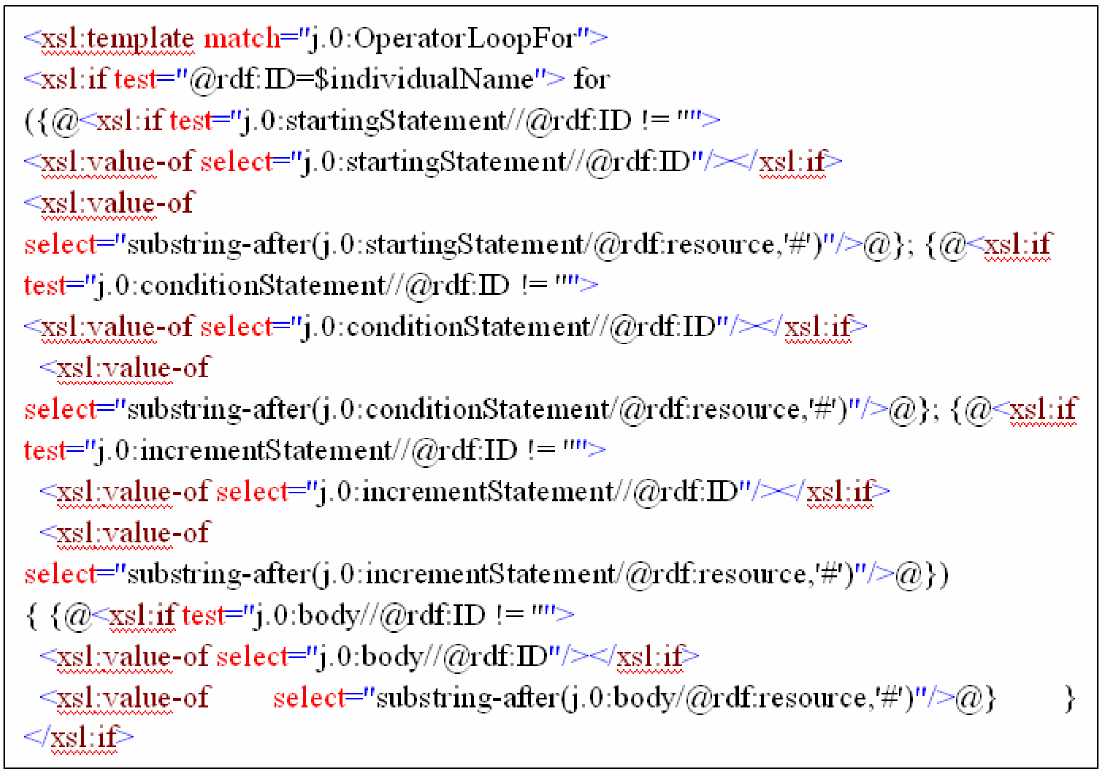
while ({@whileCondition@}) {{@ body @}}

About the Java SSLO, the XSL to transform OperatorLoopDowhile and OperatorLoopWhileDo are different from the PHP version. We had to add the 'do' string in front of "j.0:body", because Java syntax is different from PHP syntax. The final generated Java code for a OperatorLoopWhileDo individual looks like:

while ({@whileCondition@}) do {{@ body @}}

6.1.3 XSL implementation for Class OperatorLoopFor

Class OperatorLoopFor is defined in ontological algorithm to model 'for loop'



```
<xsl:template match="j.0:OperatorLoopFor">
<xsl:if test="@rdf:ID=$individualName"> for
({@<xsl:if test="j.0:startingStatement//@rdf:ID != ''">
<xsl:value-of select="j.0:startingStatement//@rdf:ID"/></xsl:if>
<xsl:value-of
select="substring-after(j.0:startingStatement/@rdf:resource,'#')"/>@}; {@<xsl:if
test="j.0:conditionStatement//@rdf:ID != ''">
<xsl:value-of select="j.0:conditionStatement//@rdf:ID"/></xsl:if>
  <xsl:value-of
select="substring-after(j.0:conditionStatement/@rdf:resource,'#')"/>@}; {@<xsl:if
test="j.0:incrementStatement//@rdf:ID != ''">
  <xsl:value-of select="j.0:incrementStatement//@rdf:ID"/></xsl:if>
  <xsl:value-of
select="substring-after(j.0:incrementStatement/@rdf:resource,'#')"/>@})
{ {@<xsl:if test="j.0:body//@rdf:ID != ''">
  <xsl:value-of select="j.0:body//@rdf:ID"/></xsl:if>
  <xsl:value-of      select="substring-after(j.0:body/@rdf:resource,'#')"/>@}       }
</xsl:if>
```

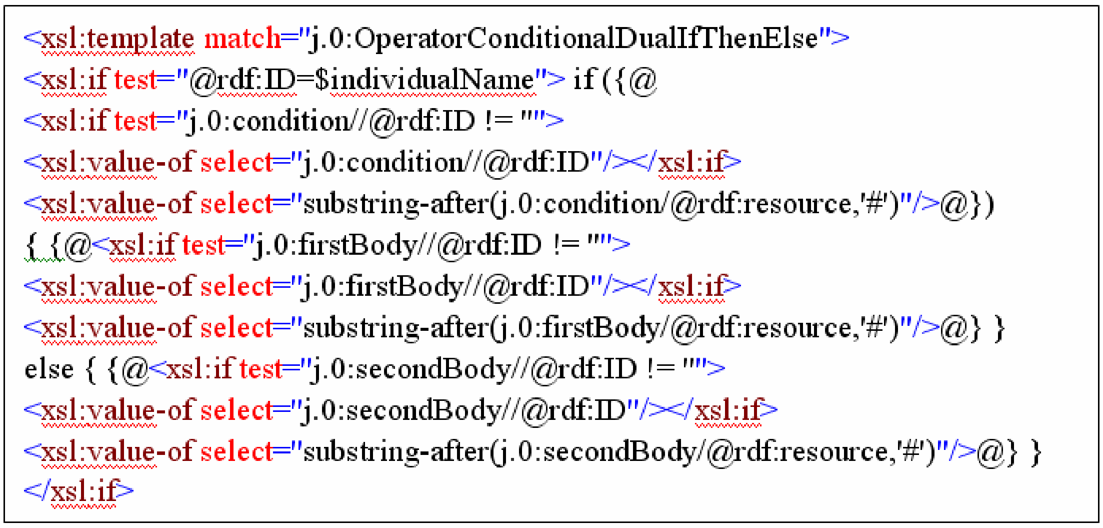
Figure 6-3: XSL implementation for Class OperatorLoopFor

As we can see from the above XSL piece of code in Figure 6-3 that the XSL template is created to search for all individuals of type “j.0: OperatorLoopFor”. And individualName is parameter passed as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. Value of property “j.0: startingStatement” is then matched under this specific individual, and values of properties “j.0:conditionStatement”, “j.0: incrementStatement”, “j.0:body” are retrieved as well. Finally generated code for this specific OperatorLoopFor individual looks like

for ({@startingStatement @},{@ conditionStatement @},{@ incrementStatement @}) {{@ body @}}

6.1.4 XSL implementation for Class OperatorCondtionDualIfThen and Class OperatorCondtionDualIfThenElse

Class OperatorCondtionDualIfThen and OperatorCondtionDualIfThenElse are defined in algorithm to model if else control flow. The difference between the two Classes is that OperatorCondtionDualIfThen only supports if statement. OperatorCondtionDualIfThenElse not only supports if statement but also supports else statement



```
<xsl:template match="j.0:OperatorConditionalDualIfThenElse">
<xsl:if test="@rdf:ID=$individualName"> if ({@
<xsl:if test="j.0:condition//@rdf:ID != ''">
<xsl:value-of select="j.0:condition//@rdf:ID"/></xsl:if>
<xsl:value-of select="substring-after(j.0:condition/@rdf:resource,'#')"/>@})
{ {@<xsl:if test="j.0:firstBody//@rdf:ID != ''">
<xsl:value-of select="j.0:firstBody//@rdf:ID"/></xsl:if>
<xsl:value-of select="substring-after(j.0:firstBody/@rdf:resource,'#')"/>@} }
else { {@<xsl:if test="j.0:secondBody//@rdf:ID != ''">
<xsl:value-of select="j.0:secondBody//@rdf:ID"/></xsl:if>
<xsl:value-of select="substring-after(j.0:secondBody/@rdf:resource,'#')"/>@} }
</xsl:if>
```

Figure 6-4: XSL implemenation for Class OperatorCondtionalDualIfThen

As we can see from the above XSL piece of code in Figure 6-4 that the XSL template is created to search for all individuals of type "j.0:OperatorConditionalDualIfThenElse". And individualName is parameter passed in as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. Property "j.0: condition" is then matched under this specific individual and value is derived, and values of properties "j.0: firstBody", "j.0: secondBody" are all retrieved. Finally generated code for this specific OperatorConditionalDualIfThenElse individual looks like

```
if ({@condition @}) {{@ firstBody @}} else {{@ secondBody @}
```

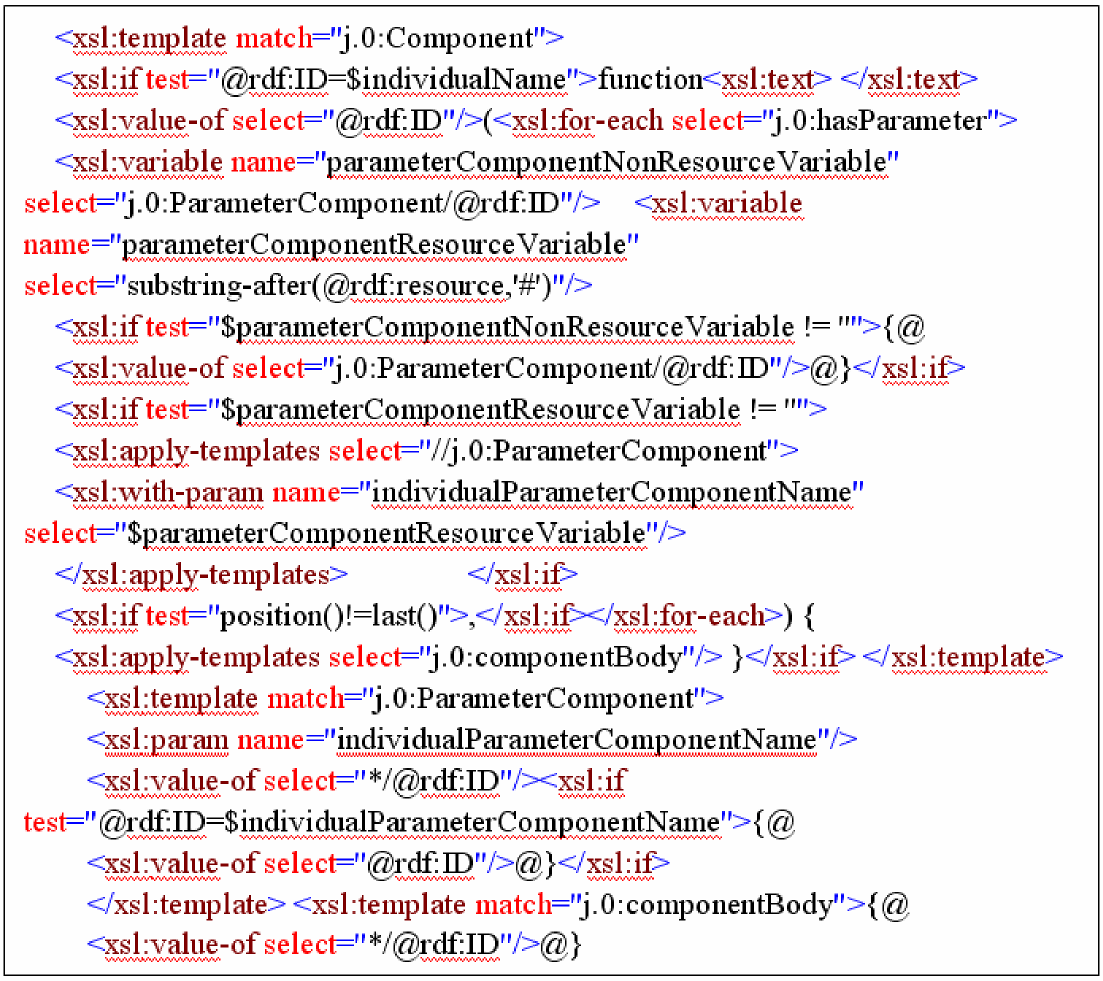
For XSL of Class OperatorCondtionDualIfThen, there is no "j.0: secondBody" property, so that it generated code looks like

```
if ({@condition @}) {{@ firstBody @}}
```

6.1.5 XSL implementation for Class Component

Class Component is defined in algorithm to model a function. XSL used in PHP SSLO and Java SSLO are different because of language syntax. In PHP, a function can be expressed as "function FunctionName{FunctionBody}" but in Java, you need to specify the function signature such as return type, public or private and you need to declare the parameter type as well.

Function in Java express like "public void functionName(String testString, int testInteger) { }"



```
   <xsl:template match="j.0:Component">
   <xsl:if test="@rdf:ID=$individualName">function<xsl:text> </xsl:text>
   <xsl:value-of select="@rdf:ID"/>(<xsl:for-each select="j.0:hasParameter">
   <xsl:variable name="parameterComponentNonResourceVariable"
select="j.0:ParameterComponent/@rdf:ID"/>    <xsl:variable
name="parameterComponentResourceVariable"
select="substring-after(@rdf:resource,'#')"/>
   <xsl:if test="$parameterComponentNonResourceVariable != ''">{@
   <xsl:value-of select="j.0:ParameterComponent/@rdf:ID"/>@}</xsl:if>
   <xsl:if test="$parameterComponentResourceVariable != ''">
   <xsl:apply-templates select="//j.0:ParameterComponent">
   <xsl:with-param name="individualParameterComponentName"
select="$parameterComponentResourceVariable"/>
   </xsl:apply-templates>            </xsl:if>
   <xsl:if test="position()!=last()">,</xsl:if></xsl:for-each>) {
   <xsl:apply-templates select="j.0:componentBody"/> }</xsl:if> </xsl:template>
      <xsl:template match="j.0:ParameterComponent">
      <xsl:param name="individualParameterComponentName"/>
      <xsl:value-of select="*/@rdf:ID"/><xsl:if
test="@rdf:ID=$individualParameterComponentName">{@
      <xsl:value-of select="@rdf:ID"/>@}</xsl:if>
      </xsl:template> <xsl:template match="j.0:componentBody">{@
      <xsl:value-of select="*/@rdf:ID"/>@}
```

Figure 6-5: XSL implementation for Class Component

As we can see from the above XSL piece of code in Figure 6-5 for PHP that the XSL template is created to search for all individuals of type “j.0: Component”. And individualName is parameter passed in as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. “Property j.0:ParameterComponent” is then matched under this specific individual to derive all the parameters of one function one by another, and then property “j.0: componentBody” is matched and its associate value is retrieved to add body part of a function. Finally generated (PHP) code for this specific Component individual looks like

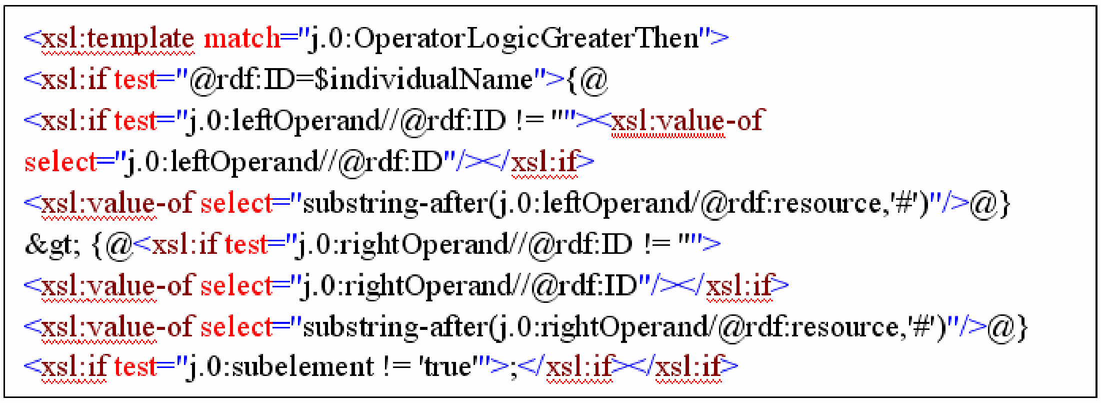
```
function FunctionName ({@parametervariable1@}, {@ parametervariable2@})
{ {@body@}}
```

For Java SSLO, XSL for transform Component is quite different that there is not “function” to be add in front of FunctionName, instead “j:0:componentSignature” property is match to retrieve its value for function signature. Also for each of the parameter of function, XSL is written to retrieve the parameter type for each of them. The final generated (Java) code for a Component individual looks like

```
public void FunctionName (int {@parametervariable1@}, float {@
parametervariable2@}) {{@body@}}
```

6.1.6 XSL implementation for Class OperatorLogicX

Class OperatorLogicEqualThen, OperatorLogicGreaterEqualThen, OperatorLogicGreaterThen, OperatorLogicLowerEqualThen, OperatorLogicLowerThen, OperatorLogicNotEqualThen are defined in ontological algorithm to model logic operator such as A>B, A<B, A<=B, etc.



```
<xsl:template match="j.0:OperatorLogicGreaterThen">
<xsl:if test="@rdf:ID=$individualName">{@
<xsl:if test="j.0:leftOperand//@rdf:ID != ''"><xsl:value-of
select="j.0:leftOperand//@rdf:ID"/></xsl:if>
<xsl:value-of select="substring-after(j.0:leftOperand/@rdf:resource,'#')"/>@}
&gt; {@<xsl:if test="j.0:rightOperand//@rdf:ID != ''">
<xsl:value-of select="j.0:rightOperand//@rdf:ID"/></xsl:if>
<xsl:value-of select="substring-after(j.0:rightOperand/@rdf:resource,'#')"/>@}
<xsl:if test="j.0:subelement != 'true'">;</xsl:if></xsl:if>
```

Figure 6-6: XSL implementation for Class OperatorLogicEqualThen

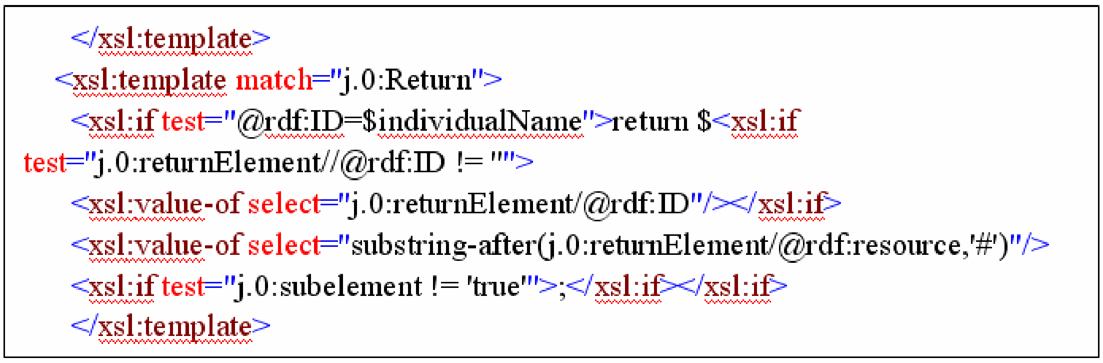
As we can see from the above XSL piece of code in Figure 6-6 that the XSL template is created to search for all individuals of type “j.0: OperatorLogicGreaterThen”. And individualName is parameter passed as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. “j.0:leftOperand” and “j.0:rightOperand” properties are then matched under this specific individual to derive the values of each property. Symbol "&gt;" in the XSL represents symbol ">" in the generated code.

Finally the generated code for this specific OperatorLogicGreaterThen individual looks like

{@leftOperand@} > {@rightOperand@}

6.1.7 XSL implementation for Class Return

Class Return is defined in algorithm to model return of a function.



```
    </xsl:template>
   <xsl:template match="j.0:Return">
    <xsl:if test="@rdf:ID=$individualName">return $<xsl:if
test="j.0:returnElement//@rdf:ID != ''">
    <xsl:value-of select="j.0:returnElement/@rdf:ID"/></xsl:if>
    <xsl:value-of select="substring-after(j.0:returnElement/@rdf:resource,'#')"/>
    <xsl:if test="j.0:subelement != 'true'">;</xsl:if></xsl:if>
    </xsl:template>
```

Figure 6-7: XSL implementation for Class **Return**

As we can see from the above XSL piece of code in Figure 6-7 that the XSL template is created to search for all individuals of type "j.0:Return". And individualName is parameter passed as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. "j.0: returnElement" property is then matched under this specific individual to derive its value representing the return variable.

Finally the generated code for this specific Return individual looks like return

${@return_variable@}

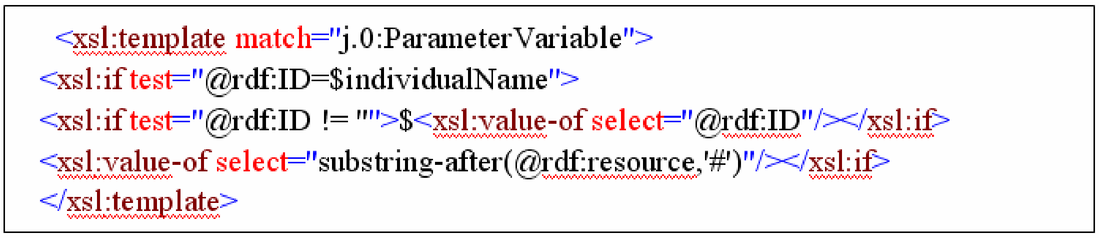
For Java SSLO, XSL for transform Return is different that "$" is removed because of Java syntax not allow it.

The final generated (Java) code for a Return individual looks like

return {@return_variable@}

6.1.8 XSL implementation for Class ParameterVariable

Class Variable is defined in algorithm to model a variable



```
  <xsl:template match="j.0:ParameterVariable">
<xsl:if test="@rdf:ID=$individualName">
<xsl:if test="@rdf:ID != ''">$<xsl:value-of select="@rdf:ID"/></xsl:if>
<xsl:value-of select="substring-after(@rdf:resource,'#')"/></xsl:if>
</xsl:template>
```

Figure 6-8: XSL implementation for Class **ParameterVariable**

As we can see from the above XSL piece of code in Figure 6-8 that the XSL template is created to search for all individuals of type “j.0: ParameterVariable”. And individualName is parameter passed as input by the Saxon XSLT Processor to specify which individual is going to be transformed among all found individuals. “rdf:resource” property is then matched under this specific individual to derive its value representing the variable. Finally the generated code for this specific Variable individual looks like

${@variable@}

For Java SSLO, XSL for transform Return is different that “$” is removed because of Java syntax not allow. The final generated (Java) code for a Return individual looks like

{@variable@}

6.2 Code Generation Implementation

This section is providing the details explanation for implementation of various components for code generation and deployment. As defined in the requirements, code is generated by interpreting the AO using SSLO. The generated code is then fulfilled to be complete PHP and Java file, and deployed to Apache web server folder. In my design that component responsible for PHP and java file deployment is separated from CGE. CGE is only responsible for generate code based on AO and SSLO.

6.2.1 CGE Classes Architecture

In CGE, six classes and one interface were created. Each of them is designed to handle different tasks in order to make code generation work.

Class OWLTreeRecursiveParser contains various recursive functions for AO tree parsing.

Class StringRegular is responsible for special pattern {@ @} matching using regular expression.

Class PHPSearchMatchedIndividual and JavaSearchMatchedInvidual inheritance from Interface SearchMatchedIndividualInterface, Here I take class PHPSearchMatchedIndividual as an example that PHPSearchMatchedIndividual is created for finding the matched XSL syntax rule in PHP SSLO of specific individual in AO.owl.

Class SaxonTranslation, once we retrieved XSL for one specific individual in the AO then SaxonTranslation is responsible for transformed this individual using retrieved XSL to be either PHP or Java code.

Class FileChooser2 is the graphic interface with various menu items on it for user interaction.

The Figure 6-9 is class diagram for overall system

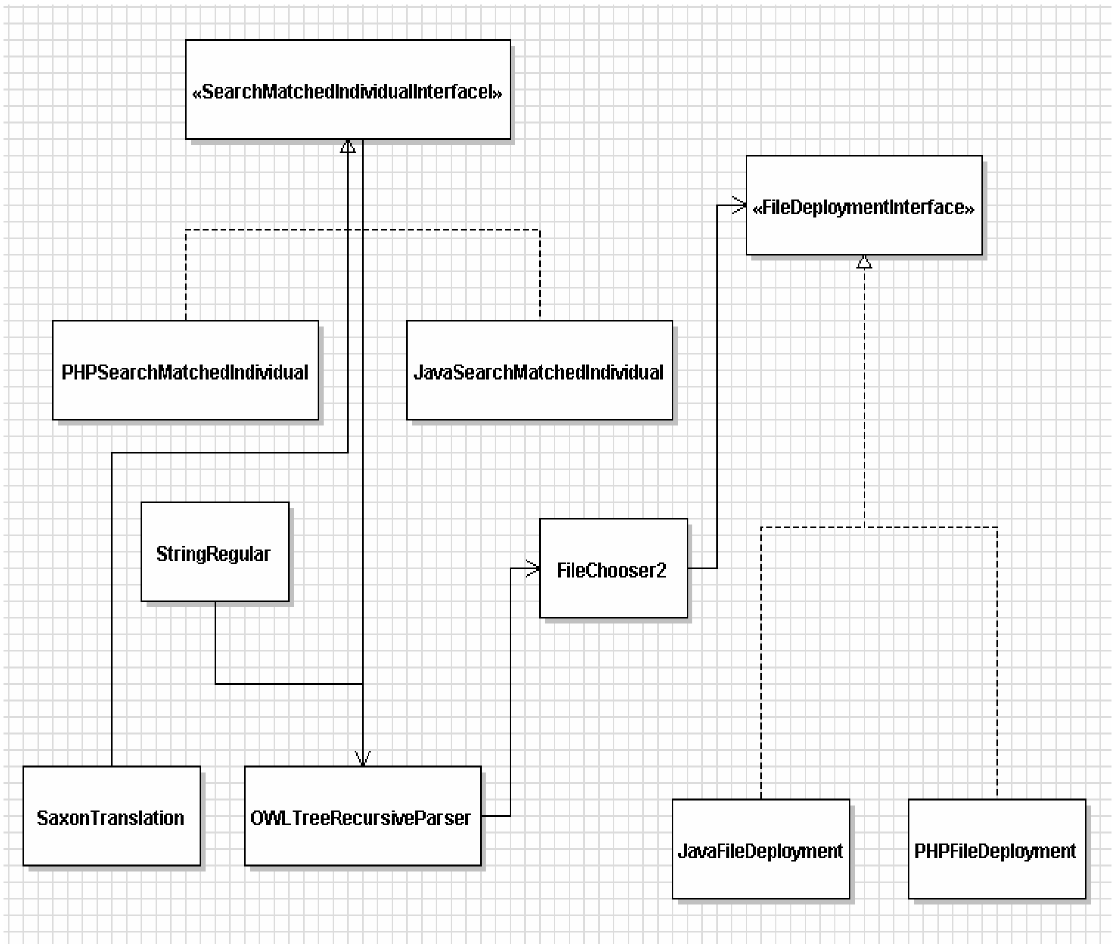

«SearchMatchedIndividualInterfaceI»
«FileDeploymentInterface»
PHPSearchMatchedIndividual
JavaSearchMatchedIndividual
StringRegular
FileChooser2
SaxonTranslation
OWLTreeRecursiveParser
JavaFileDeployment
PHPFileDeployment

Figure 6-9 CGE Class Diagram

6.2.2 Review Code Generation Processes Flow

Figure 6-10 is already explained in design chapter as code generation process flow, it mainly can be partitioned into several stages

- Find the Root Element from AO.owl and make it be to Current Element
- Translate Current Element
- Match pattern for searching SubElement(s) of Current Element
- Call recursive translation on the SubElements
- If Current Element has NextElement, call recursive translation on NextElement and make NextElement to be Current Element.
- If Current Element doesn't havee SubElement(s) to be translated and no NextElement linked to it, we reach the end of code generation.

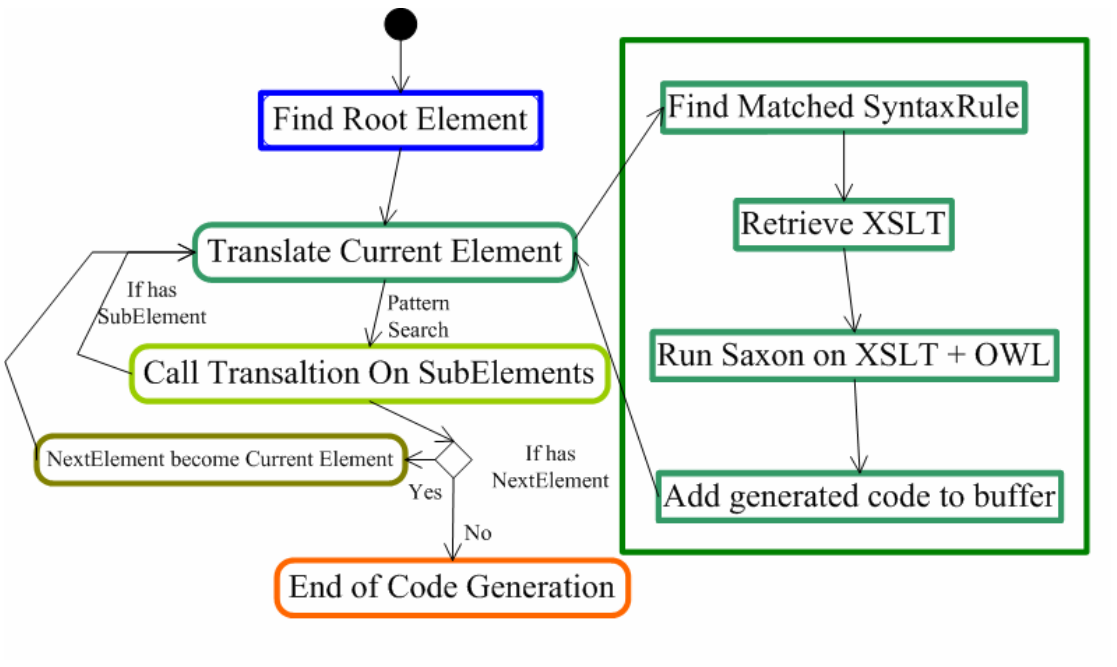
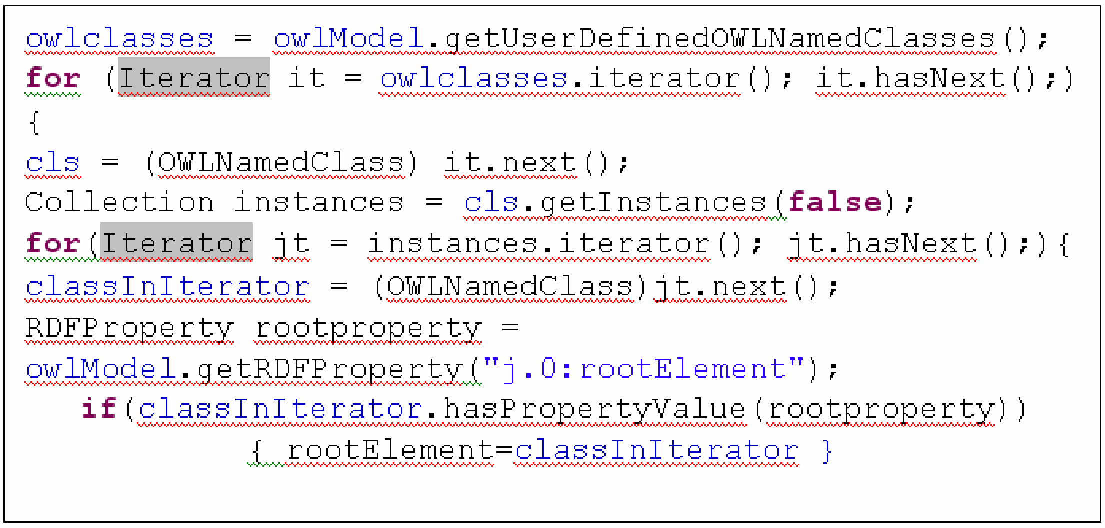


Figure 6-10: Code Generation Processes Flow

6.2.3 Implementation for finding Root Element

Finding the Root Element in AO tree is the starting point for code generation. In the OWLTreeRecursiveParser class, ParseTree() function is implemented to handle this task. Instance of Class OWLModel is created to retrieve all the classes and individuals (instances) of those classes in the AO.

From the following code shown in Figure 6-11 , it checked each individual by its “j.0:rootElement” property through iteration of all individuals. Since there is only one individual can have “j.0:rootElement” property that once we found this individual, it becomes Root Element.



```
owlclasses = owlModel.getUserDefinedOWLNamedClasses();
for (Iterator it = owlclasses.iterator(); it.hasNext();)
{
cls = (OWLNamedClass) it.next();
Collection instances = cls.getInstances(false);
for(Iterator jt = instances.iterator(); jt.hasNext();){
classInIterator = (OWLNamedClass)jt.next();
RDFProperty rootproperty =
owlModel.getRDFProperty("j.0:rootElement");
   if(classInIterator.hasPropertyValue(rootproperty))
            { rootElement=classInIterator }
```

Figure 6-11: Code Sample for finding Root Element in AO

6.2.4 Implementation for Current Element Transformation

Once Root Element is found, then it is assigned to be Current Element for transformation. Now reach the second stage of the code generation processes flow that is to transform the Current Element.

The following sequence diagram (Figure 6-12) shows the how three classes interact with each other for Current Element transformation.

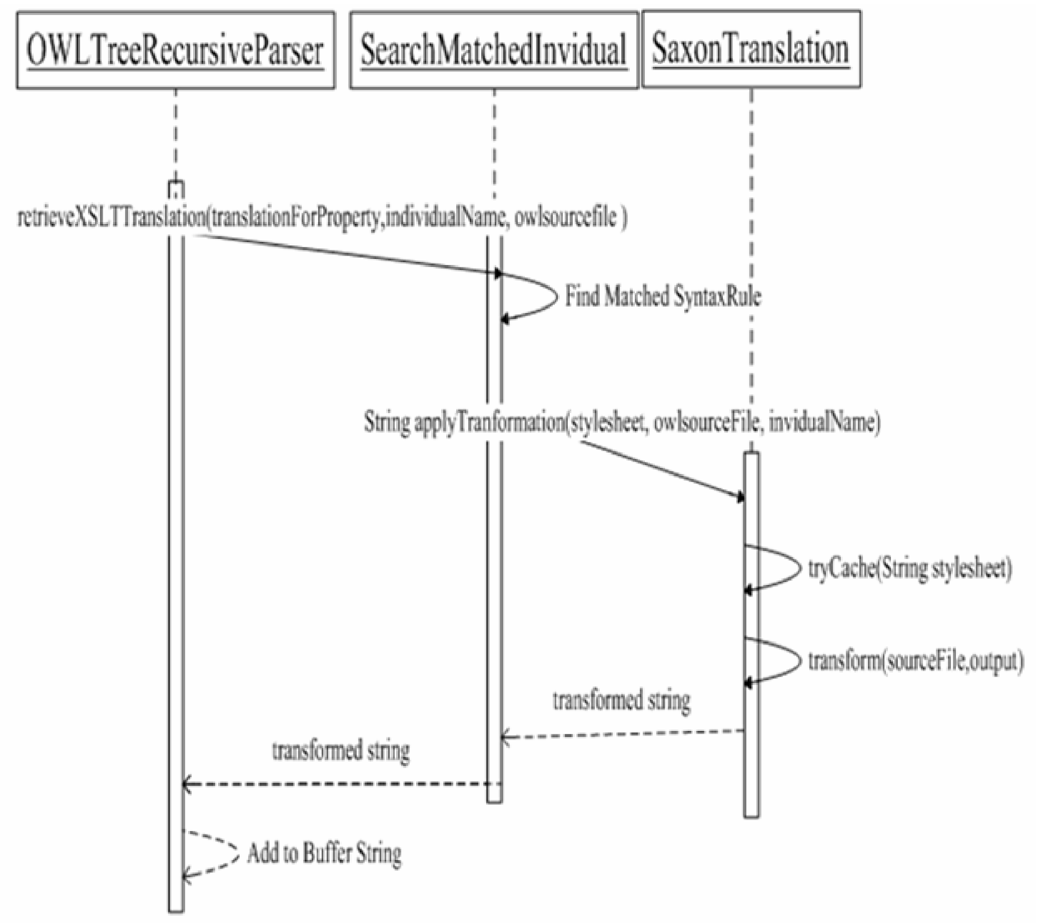

OWLTreeRecursiveParser
SearchMatchedInvidual
SaxonTranslation
retrieveXSLTTranslation(translationForProperty,individualName, owlsourcefile)
Find Matched SyntaxRule
String applyTranformation(stylesheet, owlsourceFile, invidualName)
tryCache(String stylesheet)
transform(sourceFile,output)
transformed string
transformed string
Add to Buffer String

Figure 6-12: Sequence Diagram for Current Element Transformation

Function String retrieveXSLTTranslation(translationForProperty, individualName, owlsourcefile) is then called and three parameters are passed-in. Parameter translationForProperty represents what OWL Class that current element is belong to. Parameter individualName represents name of current element, Parameter owlsourcefile represents AO file.

Inside retrieveXSLTTranslation method, it iterates through all the individuals of Class SyntaxRule. If any SyntaxRule individual property "j.0:translationFor" has the same value as passed-in parameter translationForProperty. Then we can retrieve XSL for Current Element by getting "XSLT" property value of matched SyntaxRule individual.

For example, if passed-in parameter value is "Component", it should match SyntaxRulesComponent (SyntaxRulesComponent is an individual of Class SyntaxRule). Because SyntaxRulesComponent property "j.0:translationFor" has same value as parameter translationForProperty which is “Component”.

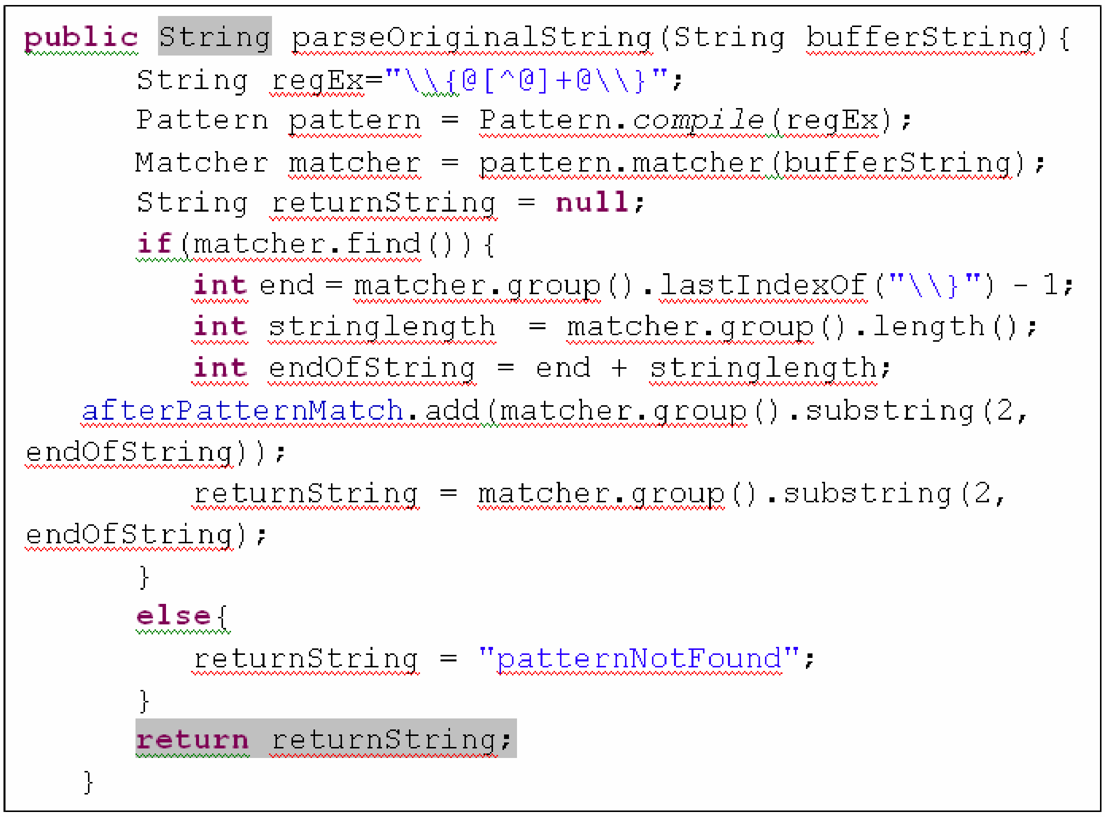
Once XSL is retrieved, next step is to call applyTransformation(stylesheet, owlsourcefile, individualName) function to get call Saxon XSLT processor to transform Current Element using retrieved XSL. Because retrieved XSL is only able to transform the Current Element, code generated after Saxon processor is only a piece of code. This piece of code is then assigned to be added in buffer string for further transformation.

6.2.5 Implementation for Special Pattern {@ @} Match

In the buffer string return from the Current Element transformation, it may include special pattern {@ @}, representing the transformation for the buffer string is not finished yet. For example, Current Element OperatorAssignEqualThan_6 can be transformed to {@variable_v@} = {@zero@}, variable_v and zero wrapped by special pattern are individuals that need to be further transformed.

Method parseOriginalString(String originalString) of Class StringRegular is designated for checking whether buffer string contains special pattern {@ @}. In the piece of code listed in Figure 6-13, it shows how regular expression is used to match special pattern in the buffer string

By calling parseOriginalString method it will return name of first matched individual wrapped by pattern {@ @}. For example, if buffer string now is “{@variable_v@} = {@zero@}”, after parseOriginalString method is called on this buffer string then variable_v is return telling the CGE that it need to be transformed. Otherwise, if there is no pattern found in the buffer string, method will return “patternNotFound” means the buffer string is complete transformed.



```
public String parseOriginalString(String bufferString){
        String regEx="\\{@[^@]+@\\}";
        Pattern pattern = Pattern.compile(regEx);
        Matcher matcher = pattern.matcher(bufferString);
        String returnString = null;
        if(matcher.find()){
            int end = matcher.group().lastIndexOf("\\}") - 1;
            int stringlength  = matcher.group().length();
            int endOfString = end + stringlength;
    afterPatternMatch.add(matcher.group().substring(2,
endOfString));
            returnString = matcher.group().substring(2,
endOfString);
        }
        else{
            returnString = "patternNotFound";
        }
        return returnString;
    }
```

Figure 6-13: Sample Code for Special Pattern Matching

6.2.6 Implementation for Recursive Functions for traversing the AO Tree and calling transformation method on each element

Because the AO is constructed as a tree, the use of recursive parsing approach is very suitable. As already mentioned in the design chapter, the elements part of the tree can be divided into three types as Current Element, SubElement(s) and NextElement. Each Current Element can have zero to multiple SubElement(s) and zero to one NextElement. A SubElement may has its own SubElment(s). There are two recursive methods, they were developed for traversing the AO Tree and calling transformation method on each element.

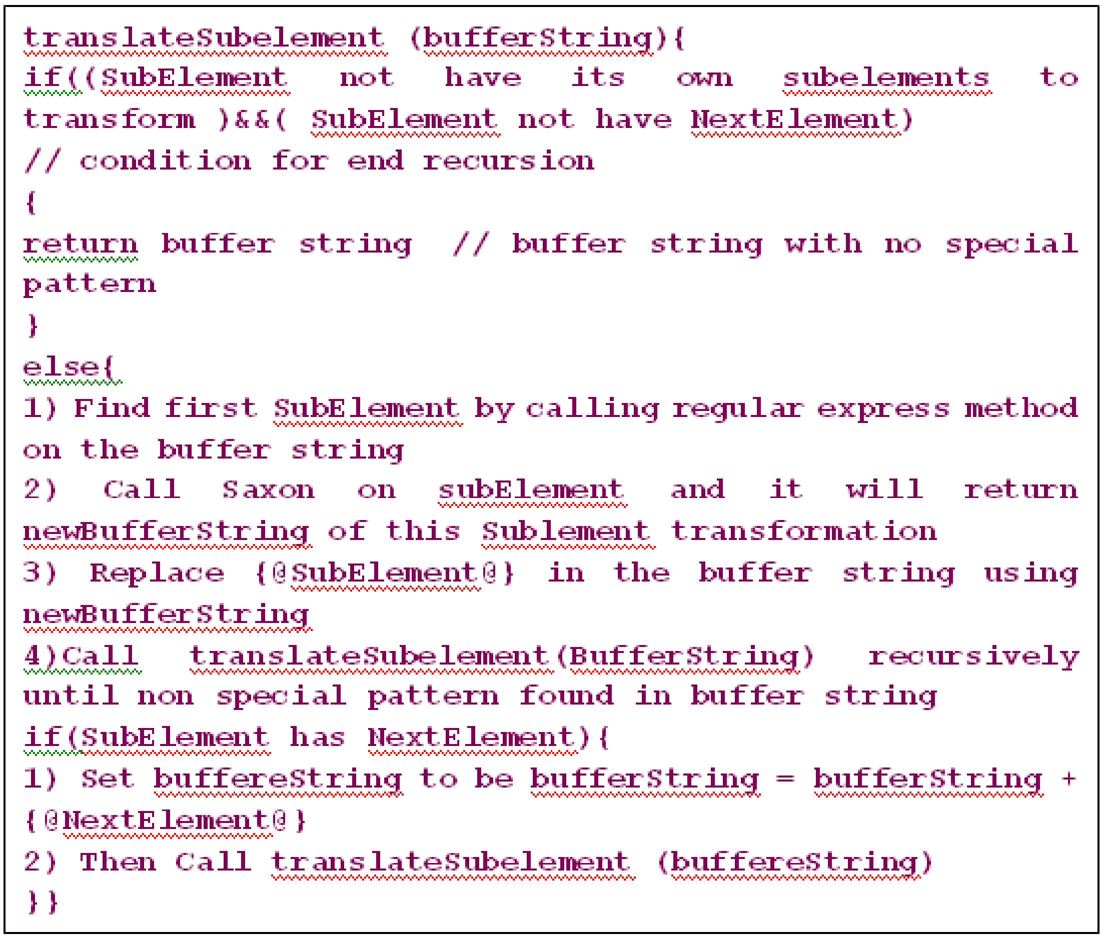
Method translateCode(String bufferString,OWLNamedClass currentElement) is responsible for traversing AO tree and call transformation method (retrieveXSLTTranslation) recursively on Current Element. The following pseudo Code (Figure 6-14) clearly explain how translateCode method works

```
translateCode(bufferString, currentElement ){
if((Current    Element    not    has    subelements    to
transform )&&(Current Element not has NextElement)
// condition for end recursion
{
return BufferString as final Output
}

else{
1) Transform the CurrentElement and get Buffer String from
Saxon XSLT Processor
2) Check whether buffer String contains special pattern {@
@}, if buffer String contains {@ @} means Current Element
has Sublement, otherwise no subelement.

if(CurrentElement has subelements){
      1)   Call translateSubelement(bufferString) method
           to recursive transform all subelement of Current
           Element, until no special pattern on the buffer
           String
      2)   Replace original buffer string(after current
           element  transformation)  using  non-special
           pattern string return from translateSubelement
           method}
      if (CurrentElement has NextElement){
      1) Add Next Element to end of bufferestring as
         bufferString = bufferString + {@NextElement@}
      2) Assign Next Element to be Current Element
      3) Call translateCode(bufferestring, NextElement)
         on new bufferString and NextElement, recursion
         starts
}}
```

Figure 6-14: Pseudo Code of translateCode method

Method translateSubelement (bufferString) is developed as a recursive method to call the transformation method on each Subelement of Current Element and return a full translate string back to transaleCode() method.

The following pseudo code (Figure 6-15) clearly show how the translateSubElement method works.



```
translateSubelement (bufferString){
if((SubElement not have its own subelements to transform )&&( SubElement not have NextElement)
// condition for end recursion
{
return buffer string  // buffer string with no special pattern
}
else{
1) Find first SubElement by calling regular express method on the buffer string
2) Call Saxon on subElement and it will return newBufferString of this Sublement transformation
3) Replace {@SubElement@} in the buffer string using newBufferString
4)Call translateSubelement(BufferString) recursively until non special pattern found in buffer string
if(SubElement has NextElement){
1) Set bufferestring to be bufferString = bufferString + {@NextElement@}
2) Then Call translateSubelement (bufferestring)
}}
```

Figure 6-15: Pseudo Code of translateSubelement method

Example of using translationCode() and translateSubelement() to transform a Current Element in detail

1) Call translationCode() on buffer sting “{@OperatorAssignEqualThan_6@}”

2) {@ @} pattern is found and OperatorAssignEqualThan_6 becomes Current Element

3) Retrieve XSL for OperatorAssignEqualThan_6 and run using Saxon, OperatorAssignEqualThan_6 is transformed to be {@variable_v@} = {@zero@}

4) Set {@variable_V@} = {@zero@} to be a temp buffer String,

Call translateSubelement(temp buffer String) on temp buffer String

6) {@ @} pattern is found, and variable_v becomes first Sub Element need to be transformed

7) Retrieve matched XSL for variable_v and run it with Saxon, variable_v is now transformed to be $variable_v

8) Replace {@variable_v@} with $variable_v in the temp buffer string and temp buffer string becomes $variable_v = {@zero@}

9) Recursively call translateSubelement(temp buffer String) on temp buffer string

10) Do exact same steps from 6) to 8), temp buffer string becomes $variable_v = zero;

11) Because there is no {@ @} pattern found in the bufferString, translateSubelement() recursion is finished, temp buffer string was sent back to translationCode() method and replace {@OperatorAssignEqualThan_6@} with $variable_v = zero;   now buffere string is $variable_v = zero;

12) Because OperatorAssignEqualThan_6’s property “NextElement” value is “OperatorLoopFor_8”. Add OperatorLoopFor_8 to the end of buffer String. Buffer String become $variable_v = zero; {@ OperatorLoopFor_8 @}

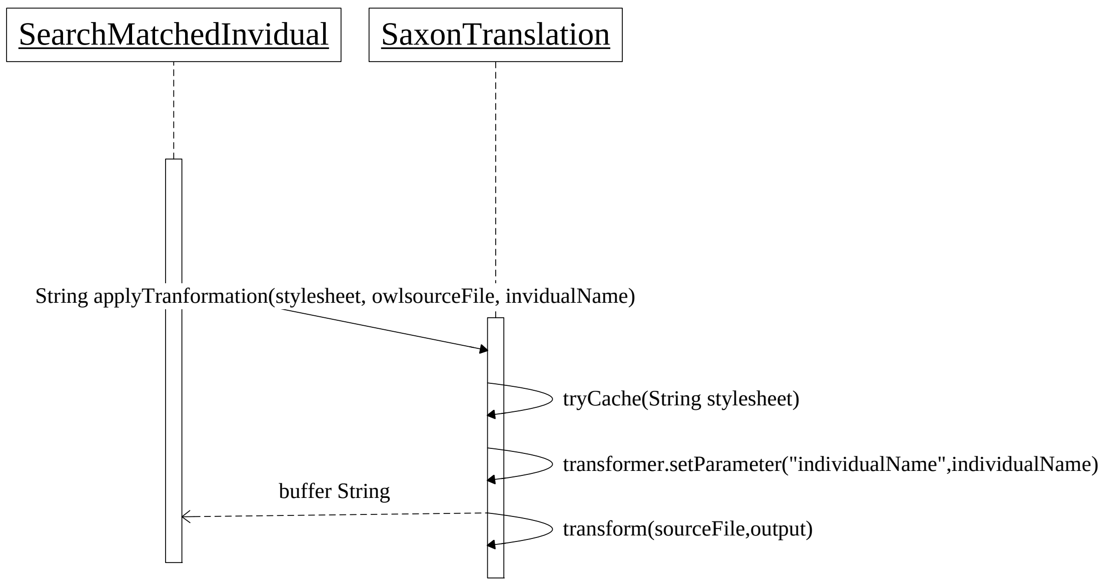
13) Call translationCode(bufferString) on new bufferString which new buffer string now is $variable_v = zero; {@OperatorLoopFor_8@}

14) OperatorLoopFor_8 becomes the new Current Element and Recursive parsing starts again.

6.2.7 Implementation for Saxon XSLT Processor

As already introduced in design chapter, Saxon API was used for handling AO and XSL transformation.

String applyTransformation(String stylesheet, File owlsourceFile, String invidualName) is major method for handling transformation. As we can see from the following sequence diagram (Figure 6-16) that three other methods are involved inside applyTransformation method.

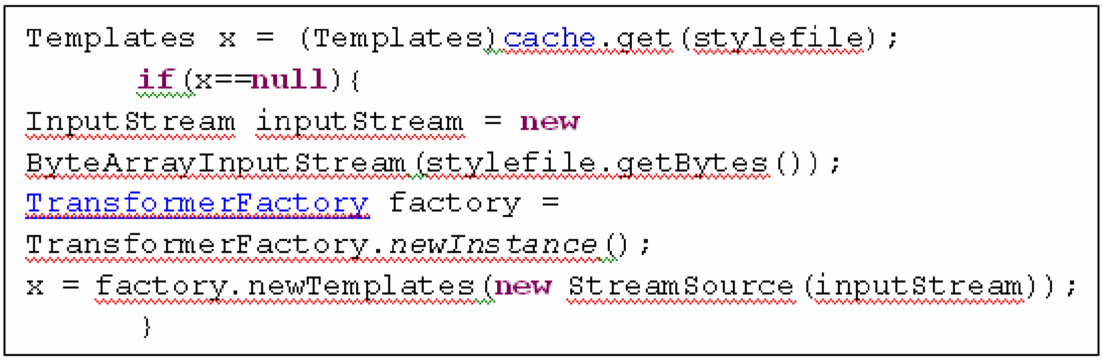


SearchMatchedInvidual
SaxonTranslation
String applyTranformation(stylesheet, owlsourceFile, invidualName)
tryCache(String stylesheet)
transformer.setParameter("individualName",individualName)
buffer String
transform(sourceFile,output)

Figure 6-16: Sequence Diagram for using Saxon for Transformation

1) tryCache(String styleSheet)

This method is to create a template for XSL as cache that it then can be used several times without recreated. XSLT TransformerFactory object was instantiated for taking XSL ready for transformation.



```
Templates x = (Templates)cache.get(stylefile);
      if(x==null){
InputStream inputStream = new
ByteArrayInputStream(stylefile.getBytes());
TransformerFactory factory =
TransformerFactory.newInstance();
x = factory.newTemplates(new StreamSource(inputStream));
      }
```

Figure 6-17: Sample Code of tryCache method

2) transformer.setParameter(“individualName”, individualName)

By call setParameter() method, the actual name of individual is set to be XSL parameter as an input parameter used in Saxon transformation.

3) transform(sourceFile,output)

This method performed the XSLT transformation AO using XSL to derived code for Current Element, output here represents the transformation result which is also known as buffer string.

6.2.8 Implementation for Code Deployment

At the moment, Code Deployment Module is separate from Code Generation Module and there are two main purposes of for implementation of this module one is making generated code to be complete PHP or Java file and anther purpose is to deploy files to web server for internet access.

1) Make PHP and Java File

Because CGE is only responsible for generated code about what AO describes, in order to make generated code to be a valid PHP or Java file, there is a need for a little additional piece of code to be added on to generated code.

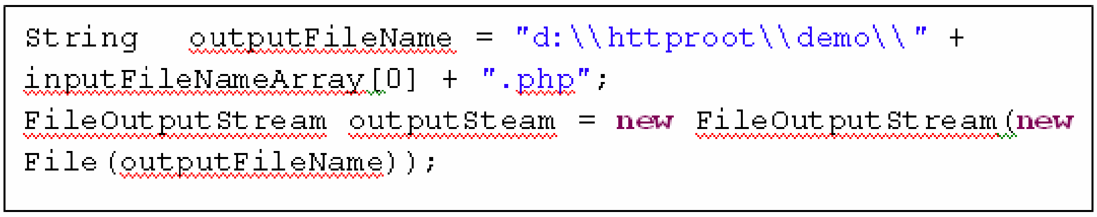
For example, PHP file needs <?php as starting of file and ?> as ending of file, AO doesn’t include those starting and ending codes at the moment. Class

PHPFileDeployment is responsible for add “<?php” in front of generated code and make “?>” append to end of the generated code.

Java file needs “public class ClassName{“ as starting of file and “}”as ending of file, AO doesn’t include this at the moment, so in Class JavaFileDeployment that it add “public class AOName{” in front of generated code and make “}” append to end of the generated code.

2) Deploy PHP and Java file to Web Server

A new FileOutputStream is created to writing PHP and Java code to specific web server folder. The list code detail explain how it achieve, and inputFileNameArray[0] represents AO filename(without“.owl”)



```
String  outputFileName = "d:\\httproot\\demo\\" +
inputFileNameArray[0] + ".php";
FileOutputStream outputSteam = new FileOutputStream(new
File(outputFileName));
```

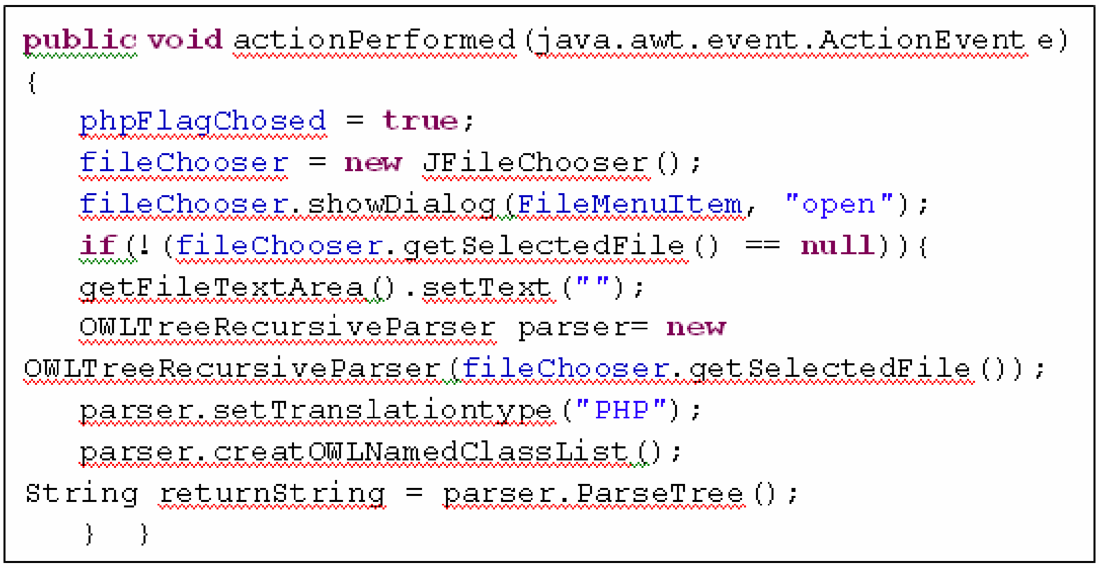
Figure 6-18: Sample Code for file deployment

6.2.9 Implementation for Graphic Interface

The Graphic Interface of this project used Java Swing to provide a user friendly widget. A set of JMenuItems are created for different purposes, JTextArea is created for showing the generated code.

JMenuItem getFileMenuItem()

The list code is event listener link to getFileMenuItem(), once this menu item is clicked, then PHP will be set to be final output language and OWLTreeRecursiveParser object will be create to starting transform AO tree, and return final generated code.



```
public void actionPerformed(java.awt.event.ActionEvent e)
{
    phpFlagChosed = true;
    fileChooser = new JFileChooser();
    fileChooser.showDialog(FileMenuItem, "open");
    if(!(fileChooser.getSelectedFile() == null)){
    getFileTextArea().setText("");
    OWLTreeRecursiveParser parser= new
OWLTreeRecursiveParser(fileChooser.getSelectedFile());
    parser.setTranslationtype("PHP");
    parser.creatOWLNamedClassList();
String returnString = parser.ParseTree();
    }  }
```

Figure 6-19: Sample Code of Action related to Menu Item

JMenuItem getJavaFileMenuItem()

This method does similar job as getFileMenuItem() except this time Java is set to be output language.

JMenuItem getMakeFileMenuItem()

This method also has a actionListener attached that once ‘Make File’ menu item is clicked, specific FileDeployment object is created to deploy the file to server folder.

Chapter 7 Evaluation

In order to evaluate the CGE, test cases were designed and tested. Several AO files were created to be used as input for CGE for specific test purposes. All those AO files are to evaluate whether CGE implemented to meet the initial aim. In this chapter, it also includes explanation in details about how to make change to AO using Protégé.

7.1 Experiment Setup

Before the actual test case starts, a few software were download and installed as experiment setup.

- The Apache Server is downloaded and installed as a experiment environment for final generated code to be run thought internet. All the Java files of CGE were packaged into a jar file acted as unique access point for testing.

- Several AO files were created for different test purposes to evaluate the CGE. CGE does only what AO describes, so the code generated from AO is valid PHP or Java code but is not sufficient to become a complete PHP and Java file. A little more information will be added to PHP and Java code in order to make them access through the internet. For example, one model of a non-recursive ascending quick sort algorithm was created for test. The code generated from this algorithm can perform the non-recursive quick sort but it still needs some additional part to add to generated code for displaying the result of non-recursive quick sort. The displaying result of non-recursive quick sort is only for convenience of test which user can see it directly on the web page.

- Protégé can be downloaded and installed to change AO if needed. For this project, all the AO files were provided by Simone to evaluate whether CGE meets his requirements.

7.2 Change AO using Protégé

This section is giving brief explain of what is Protégé and how to modify an AO using it. Some of AOs designed for test cases need to be changed to evaluate the CGE.

Basically, Protégé-OWL editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup [21] .

The changes made to the AO are very handy and straightforward in Protégé since Protégé has a very user friendly graphic interface

As we can see from the following diagram (Figure 7-1) which showing how Protégé change AO, once you open the AO and you can create an individual of defined Class and assign it to property of another, so individual is linked to another through its property. The following feature shows changes made to property ‘j.0:contion’ that OperatorAssignEqualThan_7 replace OperatorAssignEqualThan_2 to be the new value of property ‘j.0:contion’.

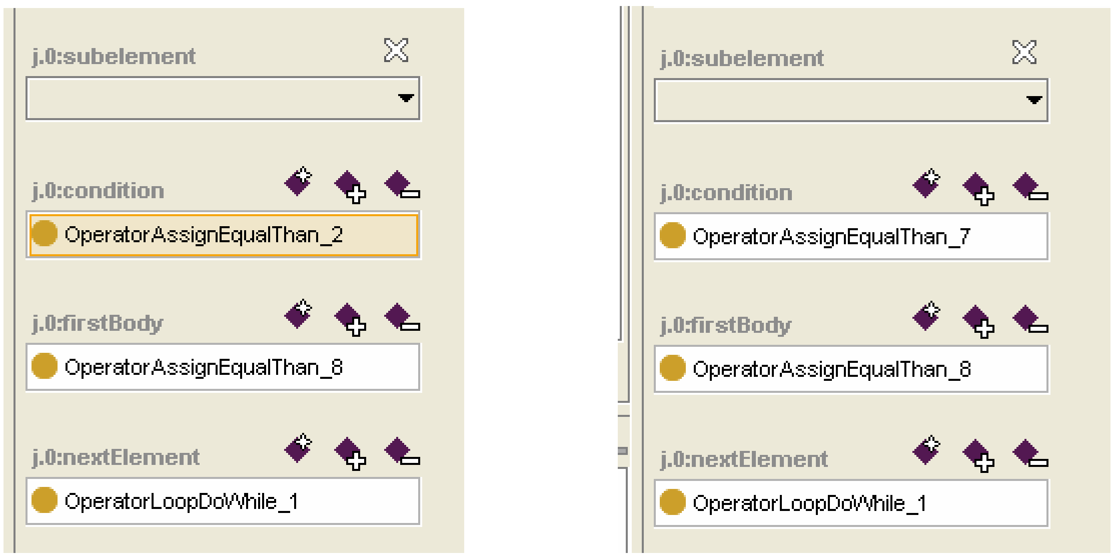


Figure 7-1: Change AO using Protege

7.3 Test Case 1

This test case it to test whether CGE is able to generate valid code by interpret AO using SSLO and generated code can be deployed and accessed through internet.

A model of a non-recursive ascending Quicksort algorithm was created based on an Array of fixed integer numbers. PHP was chosen to be the target output language. The final PHP code generated was then deployed on the server. The AO file is named as ao_nrQuicksort_v1.owl. As we can see from Figure 7-2 sthat it shows the complicate structure of this algorithm.

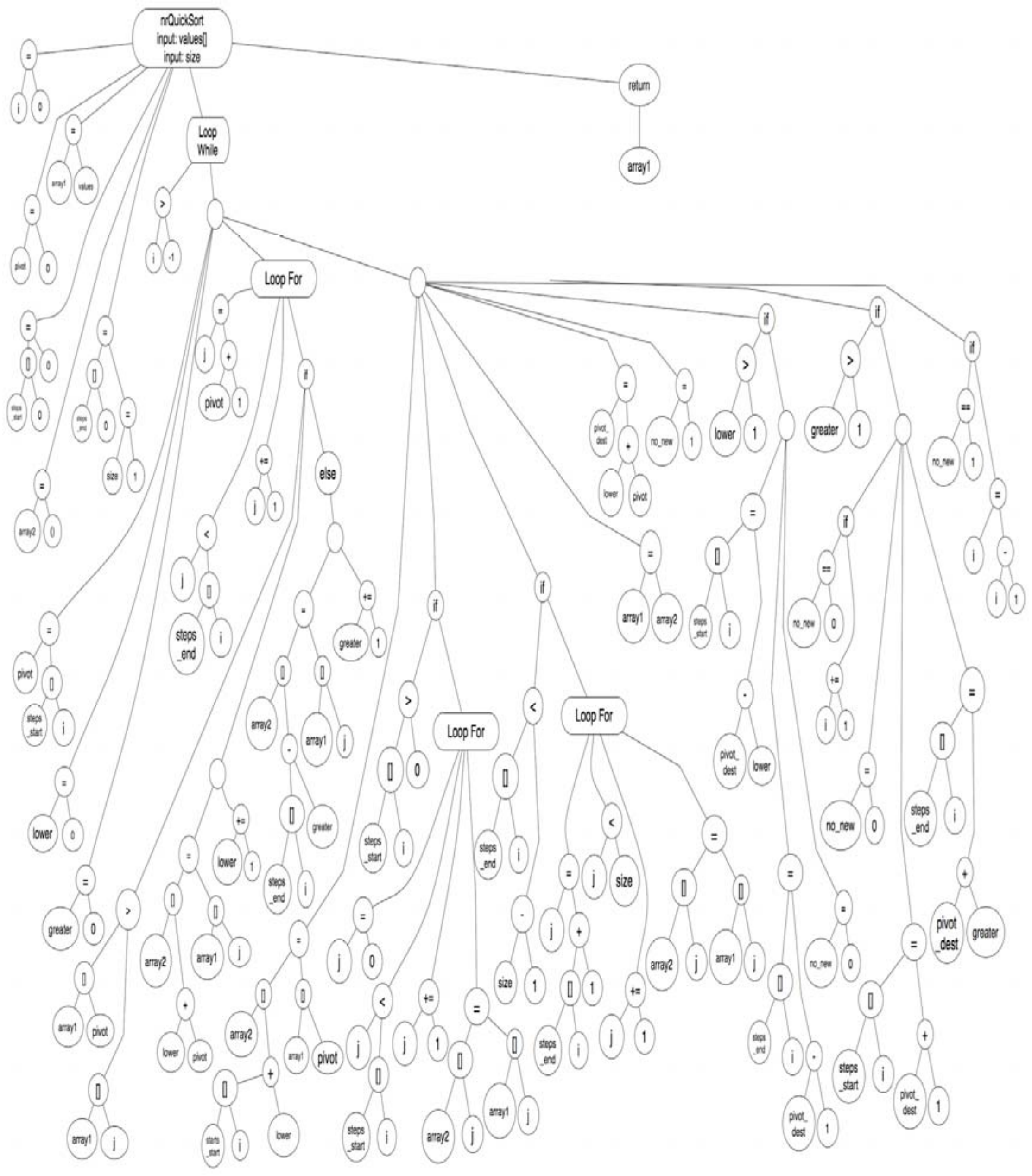
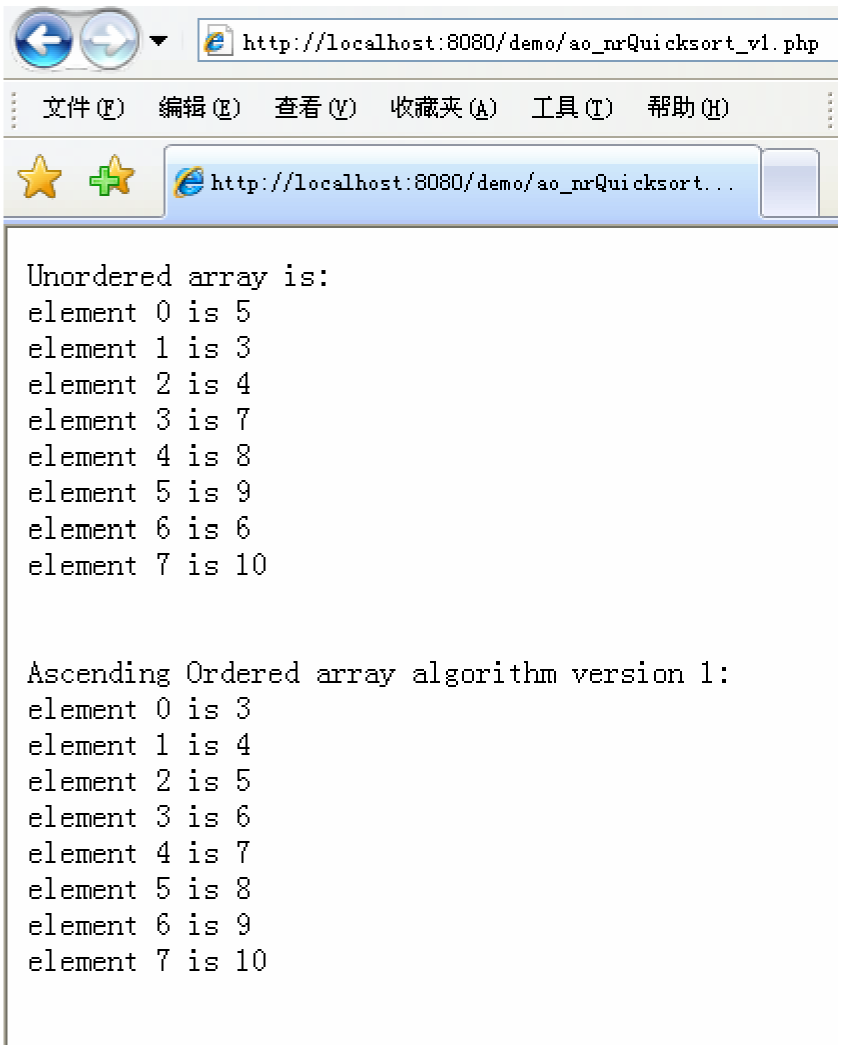


Figure 7-2: AO version1

At the starting of test, CGE selects ao_nrQuicksort_v1.owl from file system and choose PHP to be the target language. Then PHP code will be generated and display on the TextArea of CGE widget. Once the user clicks the ‘Make File’ menu item, the generated PHP code will be automatically deployed to server folder. Next step is to open web browser, the result displayed in the web browser as Figure 7-3 shows generated code does ascending non-recursive Quicksort.

This success of this test proves CGE can generate valid code as AO describes.


http://localhost:8080/demo/ao_nrQuicksort_v1.php
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
http://localhost:8080/demo/ao_nrQuicksort...

```
Unordered array is:
element 0 is 5
element 1 is 3
element 2 is 4
element 3 is 7
element 4 is 8
element 5 is 9
element 6 is 6
element 7 is 10


Ascending Ordered array algorithm version 1:
element 0 is 3
element 1 is 4
element 2 is 5
element 3 is 6
element 4 is 7
element 5 is 8
element 6 is 9
element 7 is 10
```

Figure 7-3 Snapshot of Test case 1 result in web bowser

7.3 Test Case 2

This test case 2 is to make it little to AO used in the test case 1 in order to make the ascending Quicksort algorithm to be descending Quicksort. Change was made using Protégé to ao_nrQuicksort_v1.owl and new owl is ao_nrQuicksort_v2.owl. As we can see from Figure 7-4, the tree structure of AO is not changed, only changed made to AO is the symbol “<” highlight in the diagram, it replace “>” in AO.

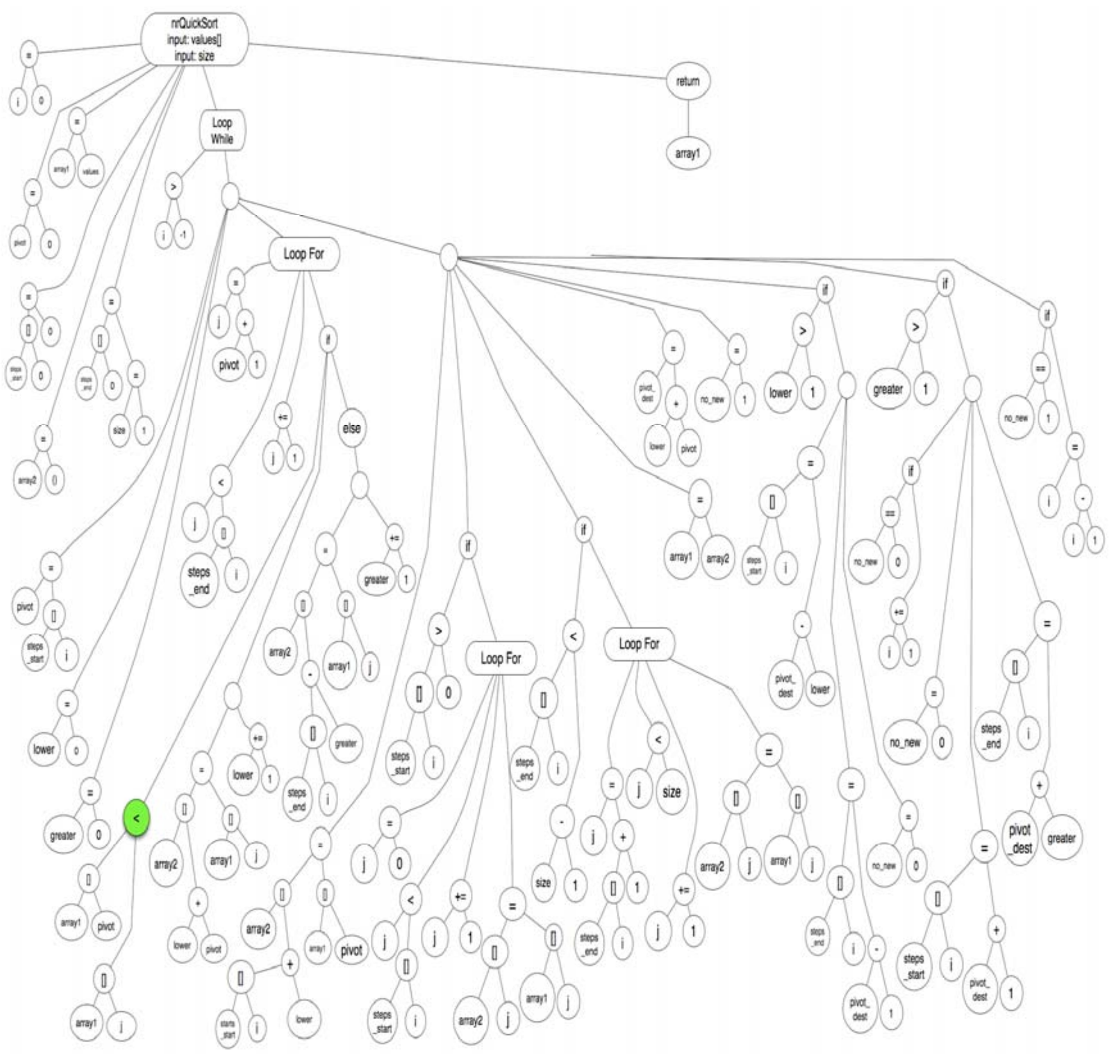
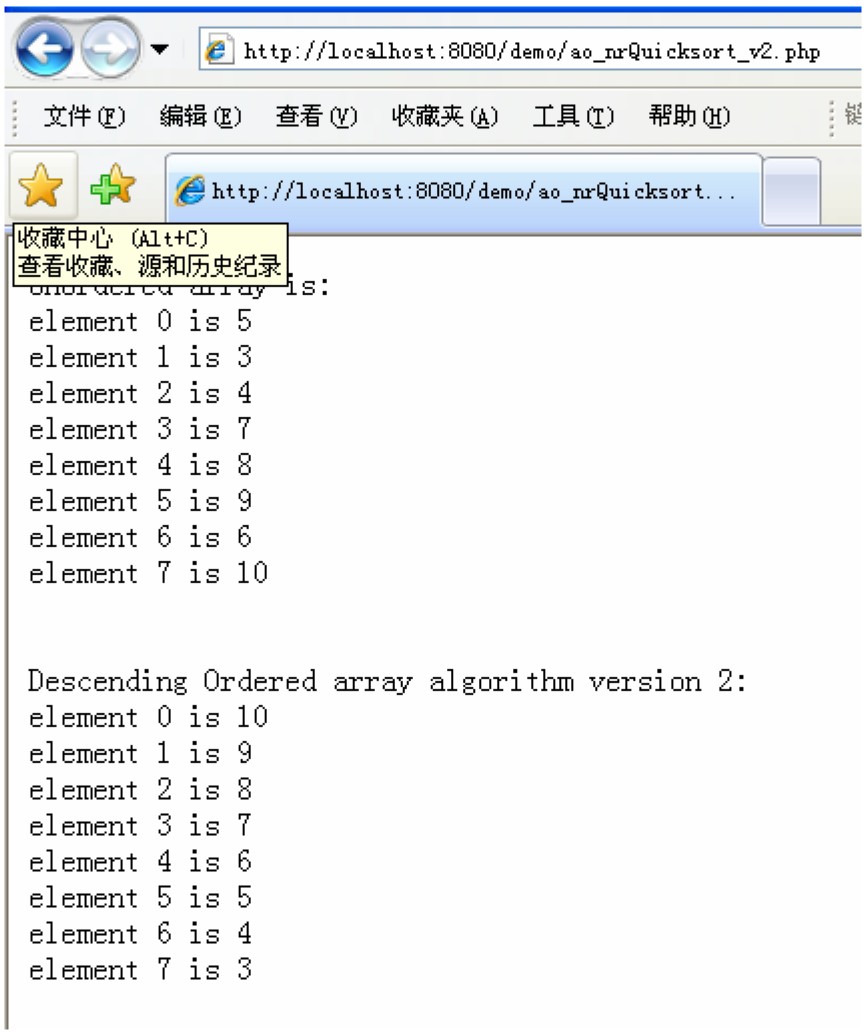


Figure 7-4: AO verion2

Then we rerun the CGE again using ao_nrQuicksort_v2.owl as input and deployed generated code, this time we can see from web browser (Figure 7-5) that change made to AO can derive changes in the final code. Ascending Quicksort algorithm becomes descending Quicksort by compare the two Figure of web browser screen cut.


http://localhost:8080/demo/ao_nrQuicksort_v2.php
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
http://localhost:8080/demo/ao_nrQuicksort...
收藏中心 (Alt+C)
查看收藏、源和历史纪录

```
Unordered array is:
element 0 is 5
element 1 is 3
element 2 is 4
element 3 is 7
element 4 is 8
element 5 is 9
element 6 is 6
element 7 is 10


Descending Ordered array algorithm version 2:
element 0 is 10
element 1 is 9
element 2 is 8
element 3 is 7
element 4 is 6
element 5 is 5
element 6 is 4
element 7 is 3
```

Figure 7-5 Snapshot of Test case 2 result in web bowser

7.3 Test Case 3

This test case 3 is to make more changes to AO used in the test case 1. The modification to AO is in order to remove all elements bigger than a specific number. Change was made using Protégé to ao_nrQuicksort_v1.owl and new owl is ao_nrQuicksort_v3.owl. As we can see from Figure 7-6, new highlight branches are added to original of the AO version1 representing actual changes in the AO file. A conditional checking is made for all the value from ascending Quicksort (AO version1). If any value is larger than specific number, then get rid of it.

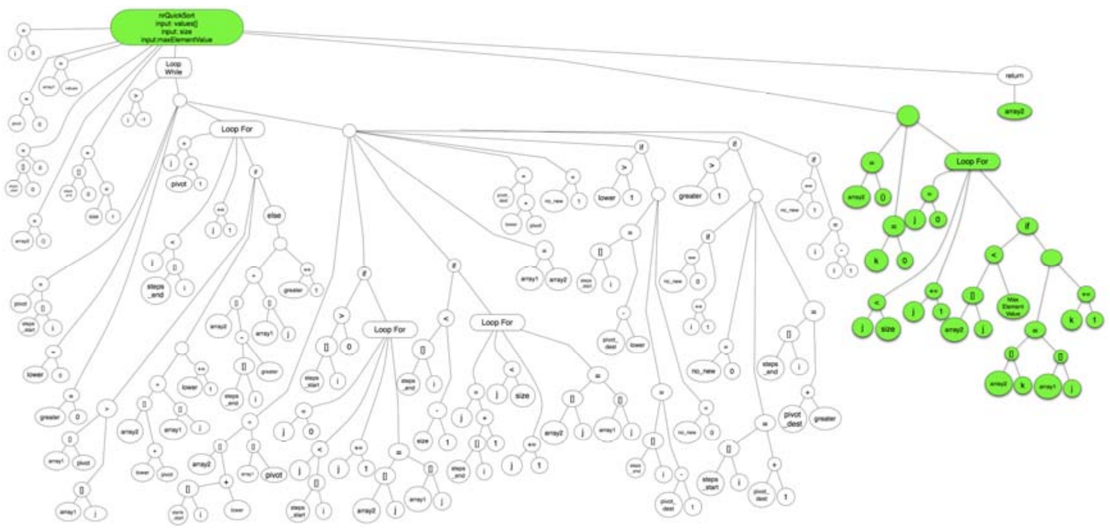
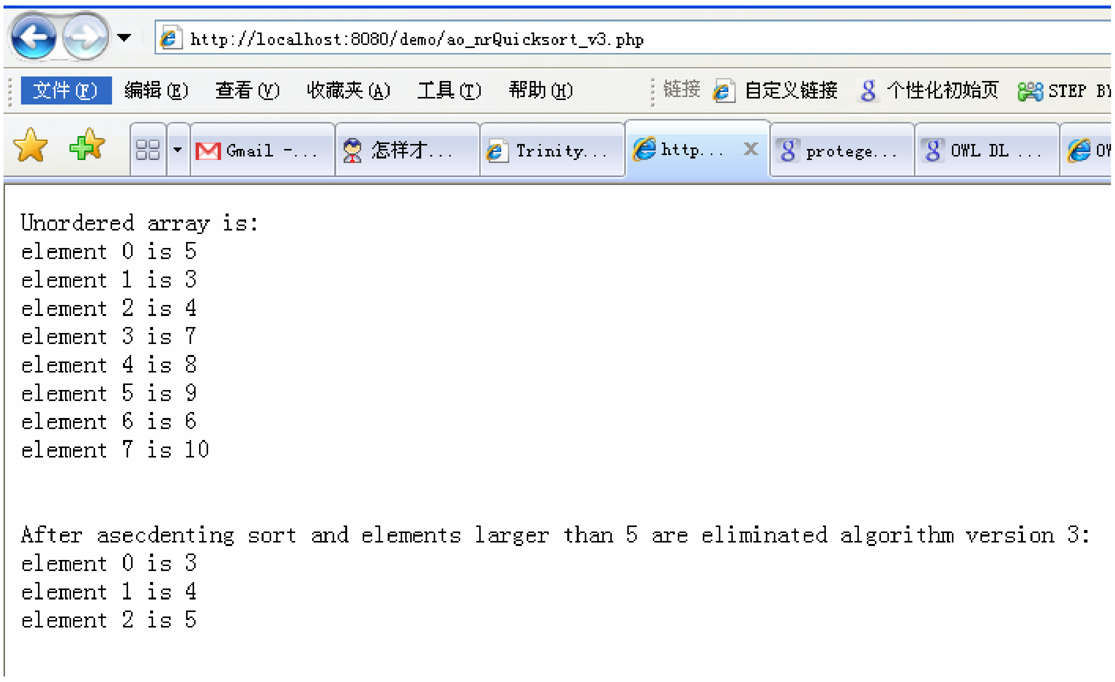


Figure 7-6: AO verion3

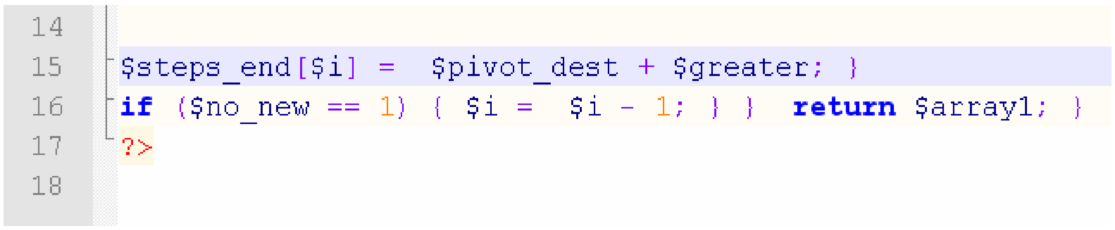
Then we rerun the CGE again using ao_nrQuicksort_v3.owl as input and deployed generated code, this time we can see from web browser (Figure 7-7) that change made to AO can derive changes in the final code. All value larger than 5 is eliminated from array by comparing the two Figure of web browser screen cut.



```
Unordered array is:
element 0 is 5
element 1 is 3
element 2 is 4
element 3 is 7
element 4 is 8
element 5 is 9
element 6 is 6
element 7 is 10


After asecdenting sort and elements larger than 5 are eliminated algorithm version 3:
element 0 is 3
element 1 is 4
element 2 is 5
```

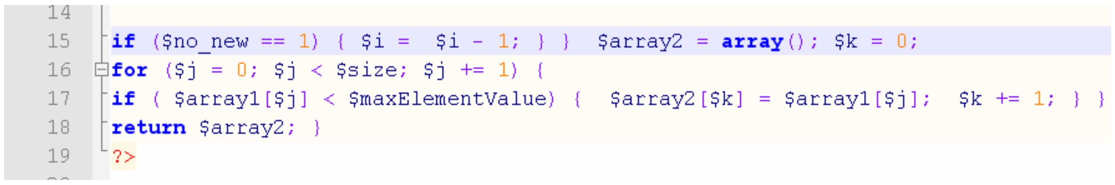
Figure 7-7 Snapshot of Test case 3 result in web bowser

Figure 7-8 and Figure 7-9 are two screen cut of actual PHP files generated by CGE. By comparing them, it is easy to see change to AO does make change to final output code.



```
$steps_end[$i] =  $pivot_dest + $greater; }
if ($no_new == 1) { $i =  $i - 1; } }  return $array1; }
?>
```

Figure 7-8 Snapshot of PHP code using Ao version1



```
if ($no_new == 1) { $i =  $i - 1; } }  $array2 = array(); $k = 0;
for ($j = 0; $j < $size; $j += 1) {
if ( $array1[$j] < $maxElementValue) {  $array2[$k] = $array1[$j];  $k += 1; } }
return $array2; }
?>
```

Figure 7-9 Snapshot of PHP code using Ao version2

7.4 Test Case 4

This test case 4 is to using a new AO which is complete different from AO we tested in previous test cases and the purpose for this AO is to evaluate whether CGE can generate Java code and PHP code do the same job based on the same AO even if they have different language syntax . The new AO structure is showing in the Figure 7-10 and new AO name is ao_AverageValue.owl.

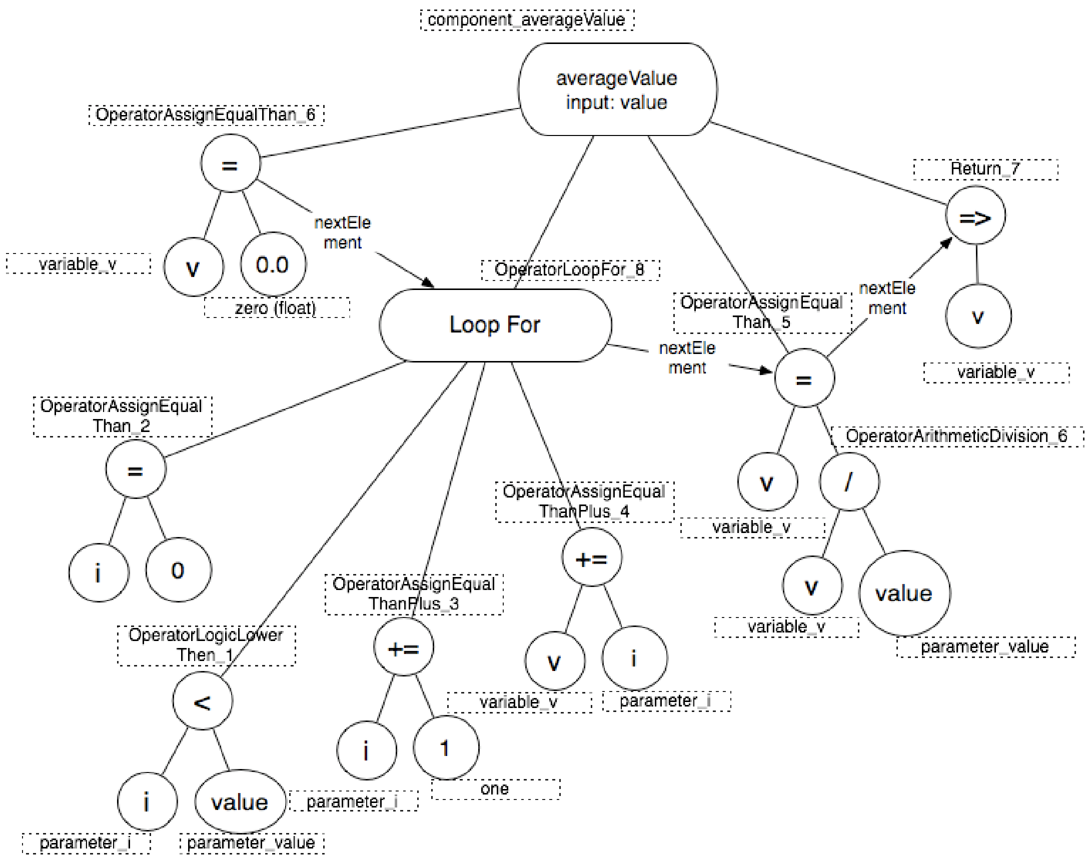
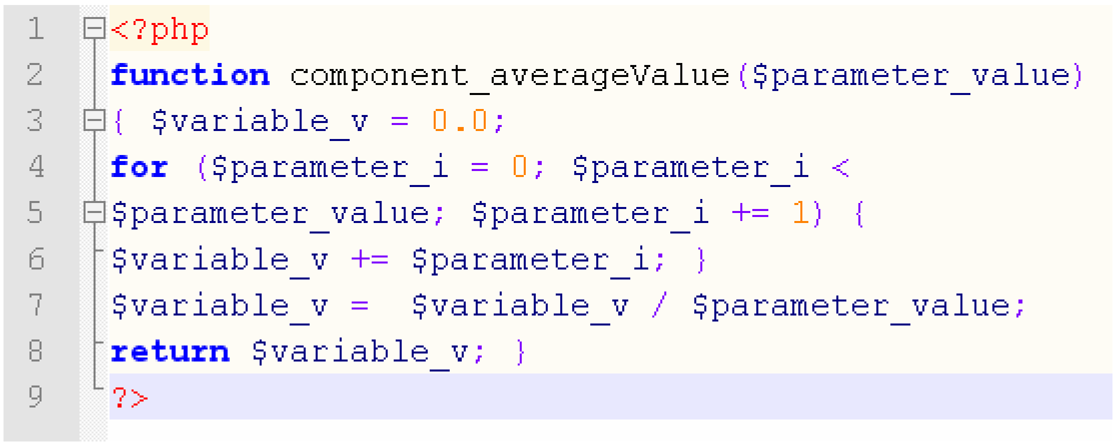


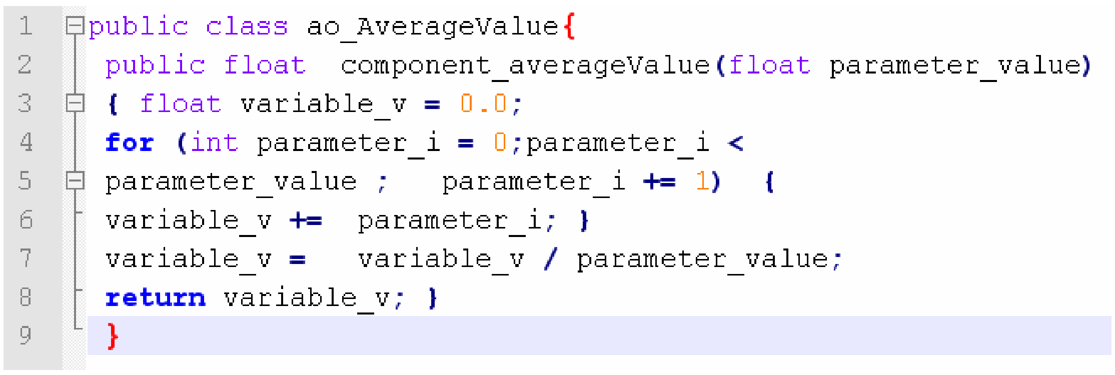
Figure 7-10 Tree Structure of ao_AverageValue AO

Then we used CGE to take ao_AverageValue.owl as input and generate both PHP and Java code, as we can see from Figure 7-11 and Figure 7-12 which are code generated in Java and PHP version based on AO, even if the language feature is different but they are doing the same jobs.



```
<?php
function component_averageValue($parameter_value)
{ $variable_v = 0.0;
for ($parameter_i = 0; $parameter_i <
$parameter_value; $parameter_i += 1) {
$variable_v += $parameter_i; }
$variable_v =  $variable_v / $parameter_value;
return $variable_v; }
?>
```

Figure 7-11 Snapshot of PHP code using ao_AverageValue AO



```
public class ao_AverageValue{
 public float  component_averageValue(float parameter_value)
 { float variable_v = 0.0;
 for (int parameter_i = 0;parameter_i <
 parameter_value ;   parameter_i += 1)   {
 variable_v +=  parameter_i; }
 variable_v =   variable_v / parameter_value;
 return variable_v; }
 }
```

Figure 7-12 Snapshot of Java code using ao_AverageValue AO

7.5 Test Case 5 based on Real Enterprise oriented Case Study

This test case is an extension of the Service Oriented Architecture Case Study presented in [19, 20], where a tax calculation system is in place and a government provide to a set of regions a base calculation algorithm. Then regions have autonomy to change the algorithm to match the need of local change in legislation.

In the first version of the algorithm the calculation of annual taxes to pay is provided, based on personal details of the person. The taxes are based on a set of layer with increasing percentage of tax to pay for each layer, than additional discounts are granted in presence of child, or in case of young age. In version 1 [Figure 7-13] there are all the parameters needed to do the calculation. A set of layers, and the percentage of tax needed to pay for each layer. Then if the person has a child get a discount, and finally the age of the person is checked, to test if it is lower than a specific parameter, to decide to assign a second discount.

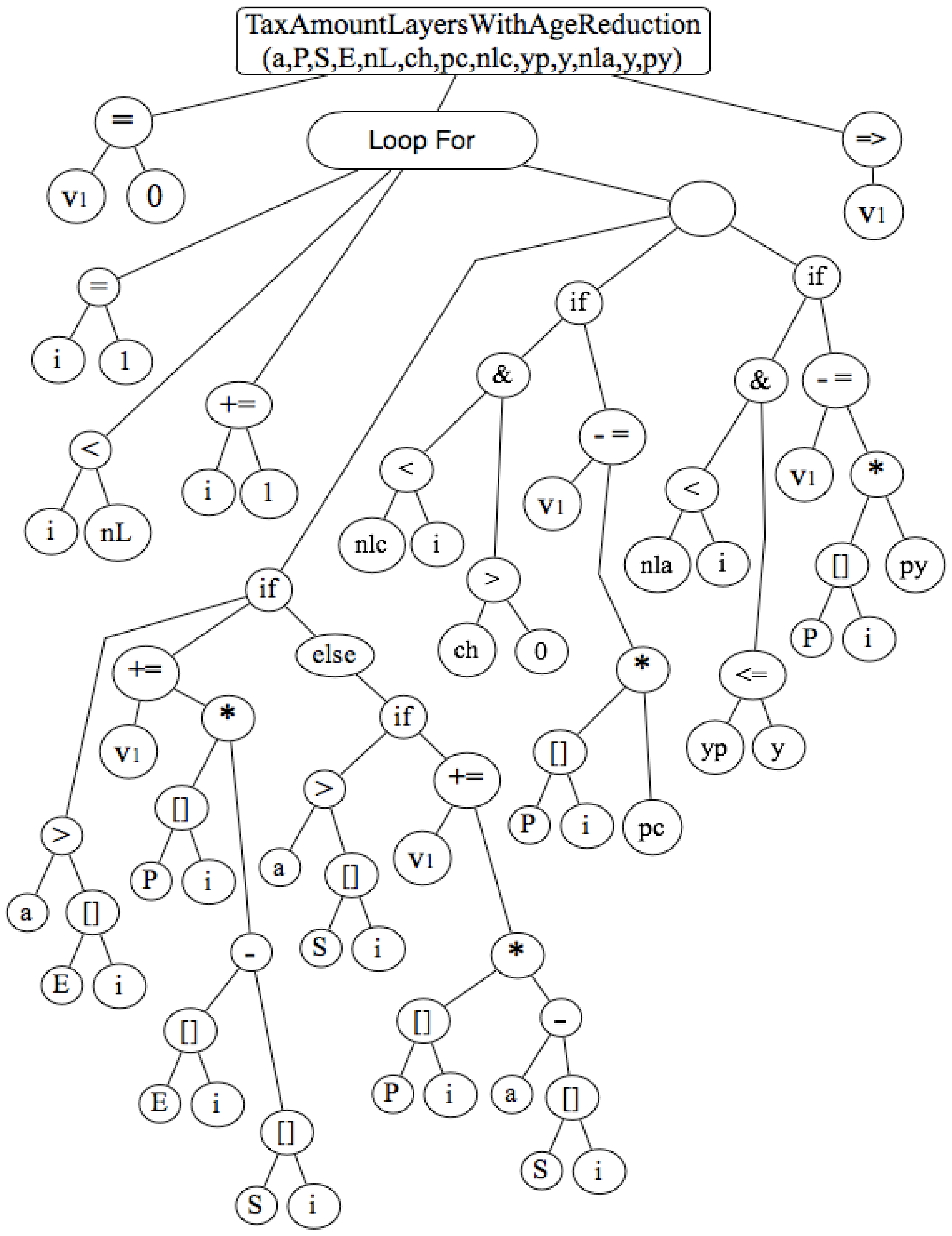

TaxAmountLayersWithAgeReduction (a,P,S,E,nL,ch,pc,nlc,yp,y,nla,y,py)
Loop For
=
v1
0
=>
i
1
<
nL
+=
if
&
nlc
>
ch
- =
*
[]
P
pc
nla
<=
yp
y
py
else
a
E
S
-

Figure 7-13 Tree Stucture of Tax AO version 1

The second version of the algorithm can be created by the central government or one of the regions, to add a discount for married people in charge of the partner. This adaptation shows how the ontological model is suitable for an adaptation than it can be sent to different systems to be transformed to working code in different frameworks based on different programming languages.

As you can see from the following graph we want now to do a change (The changes to add this modification are the highlight ones.) to the algorithm, adding an additional discount in case the person is married and a partner in charge.

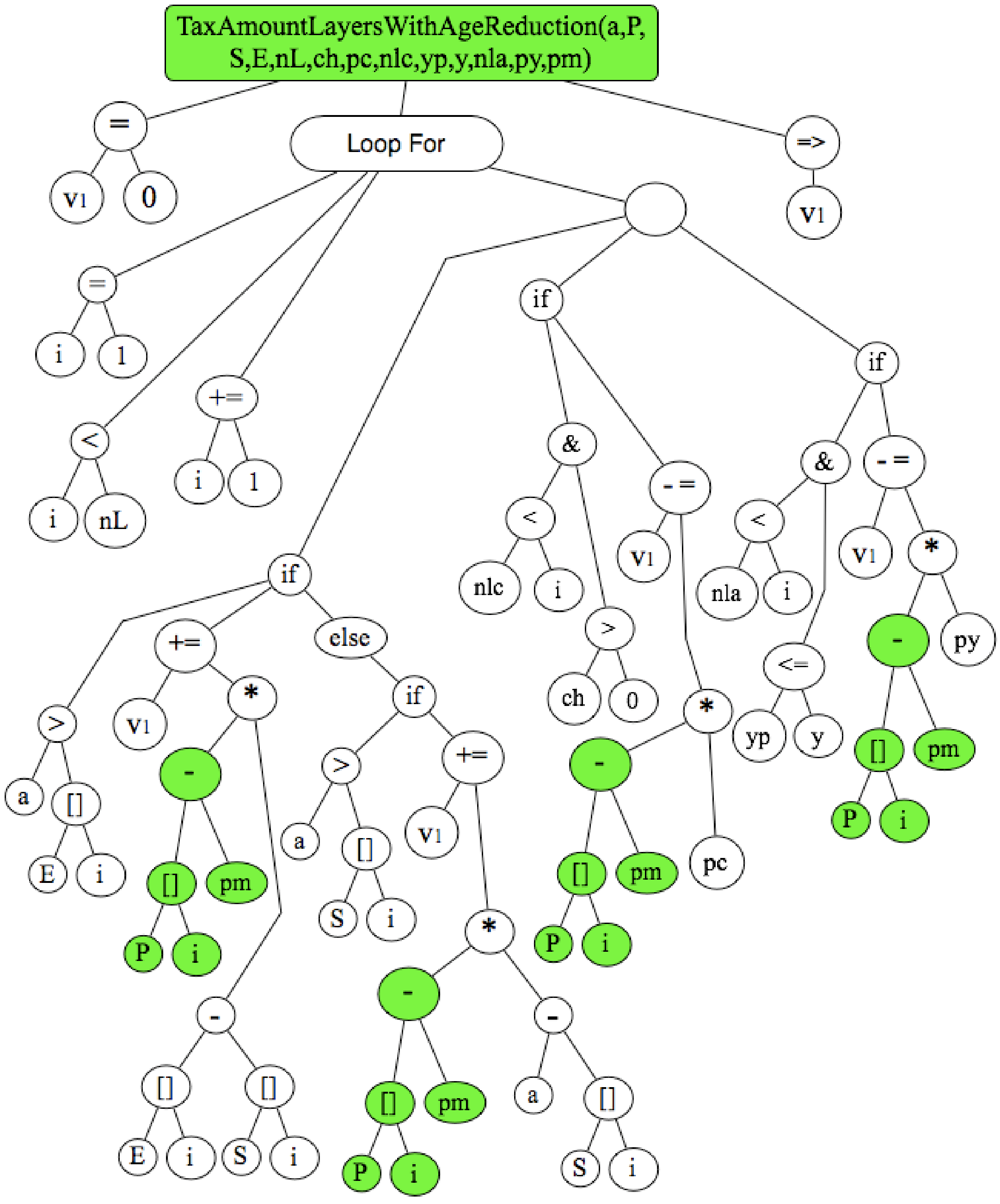

TaxAmountLayersWithAgeReduction(a,P,S,E,nL,ch,pc,nlc,yp,y,nla,py,pm)
Loop For
=
v1
0
=>
=
i
1
<
nL
+=
if
&
nlc
-=
>
ch
*
pc
-
[]
pm
P
else
a
E
S
py
nla
<=
yp
y

Figure 7-14 Tree Stucture of Tax AO version 2

The first change was to add the parameter pm (in code is $percentageMarried) to the component.

The second and last modification is to transform element 'P[i]' to element 'P[i] - pm' the element P[i] in version 1 was used 4 times but was a unique single individual.

In this case was enough to change that to 'P[i] - pm' to obtain 4 changes in one single modification. Watching the tree for version 2 (shown in Figure 7-14)is visible that the second modification spread in few places in the tree and is an aspect oriented style of adaptation, obtained with a single action in the tree instead of changing the code in 4 different parts.

After generated and tested with two version of code from the CGE, changes was made to the first version of the algorithm, to move it to the version 2. An additional discount is added in case the person is married and in charge of the partner. This complicate test Case 5 based on real Enterprise oriented Case Study further proves the power of CGE that by giving more complicate AO that CGE is able to generate code to be used in the real business for enterprise purpose.

7.6 Evaluation Summary

Based on the 5 Test Case, the CGE meets the initial requirements, it is able to generate code as what AO describes, and changes made to AO will derive changes to final generated code, the make procedure was done to test the generated code using a web browser. CGE is also able to generate two version of code (Java and PHP). A very complicate AO based on Real Enterprise oriented Case Study then created for proving CGE has its strong power in generating code to be used in the real business for enterprise purpose.

Chapter 8 Conclusion

This chapter includes project summary, influence from collaboration work and lists of contributions achieved and point out future work to be done in the further development.

8.1 Project Summary

The initial goal of this overall research project is to use abstract specification of ontology-based algorithm as Model Driven Platform to derive concrete implementation. The ontology-based modeling algorithm approach also support open set of languages by providing various language specific ontologies. The main purpose of this project is implementation of a code generation engine using ontology-based modeling algorithms.

From the state of art studies, knowledge was gained in key processes for developing a Code Generator and practice were done in get familiar with technologies to be used in the implementation. The structure of Code Generation Engine is well designed to make it stable and decouple from any particular framework or programming language. The code generated from CGE can be function, full classes or any valid code. And any change made to algorithm will lead to the change in the final code generated but no change is needed to CGE itself. Code deployment is also concerned and implemented. The Code Deployment Module is able to deploy generated code from CGE to a specific web server folder for internet access.

Various test cases were created and tested. Result of those test cases are very positive that it growth confidence for CGE to be a successful code generation tool using ontology-based modeling algorithm.

Overall project is a collaboration work on using abstract modeling algorithm to derive concrete implementation. At this stage, we can say the CGE project achieve its initial aims and becoming a successful tool for ontology-based code generation.

8.2 Influence from collaboration work

As long as this project is collaboration with another research project, it was very important to share the same research aim and have proper plan to lead to a successful development and tests. Starting April 2008 and during summer time, meetings on a weekly basis has been done. I took first month in understanding the original research project and his approach about ontology-based abstract algorithm modeling, and I did background research on the technologies which I was going to use in the future development of the sub-project. We then did partition the work that was then assigned to the main or the sub project. We agreed that the original project was in charge to refine the set of designed ontologies and to produce Algorithm Ontology, all part needed to test the architecture of the sub-project. We did agree to do a common work about the implementation of System Specific Logic Ontology, in particular to write some XSL. They were needed to transform each individual that might be used by the AO. Then it was assigned to me my major work, that was the implementation of a generic code generation engine. Finally I was in charge of the deployment procedure needed to property position the generated code on a web server document folder.

I keep updating my work status with Simone and refine the code generation engine architecture if necessary in the design stage and may come up new idea such as graphic user interface to make code generation engine more users friendly.

On August, we majorly tested different AO as input files for CGE to ensure CGE is able to meet our initial requirements. A few mistakes were found in the alpha version of CGE and a few changes were made and then test for another several rounds.

8.3 Contribution

- CGE meets Simone’s research aim that it is able to generate code using modeling algorithm approach and it is not to generate any possible programming language structure or trick, but only to generate valid code starting from the ontology that is the model of an abstract algorithm. A questioned is designed for evaluating this project from Simone’s point of view (see Appendix) and very positive feedback is achieved.
- CGE is also successful tool to test specification of abstract algorithms and it will be core part for test and evaluation part of a paper to be submitted in the future.
- CGE is very generic and it is not dependent on any particular language or framework and it is easy to extend to a new language by creating a new SSLO. And by adding new syntax rule to existing SSLO shall provide supporting more language features to handle more complicate tasks.
- By using CGE, it provides ability for supporting more dynamic and pluggable web service. User can ask for new version of web service any time by making modification to ontology algorithm and CGE can automatically derive new version of service and make it redeployed.

8.4 Future work

Possibilities of future work are discussed here:

1) Add more sophisticate parts to SSLO

At the moment, SSLO is not supporting data access, if giving more time, more attention will be paid to add more XSL rule in SSLO in providing code generation for access create query, update data in the database. This part is quite important since the fact that most of service on the web is need to process user data, and normally these data is stored in the back end data. The completion of data access rules in SSLO will provide CGE ability to generate code from front tier to back end of overall web based application.

2) Refine Java SSLO to support all Java language features

Because Java language feature is very different from PHP that is a scripting language, while Java is object-oriented language which has its unique like inheritance, polymorphism, etc. The refinement of Java SSLO should strength the ability for providing more sophisticate java code to handle complicate tasks.

3) Error Checking

CGE is responsible for generate code based on what AO describes. So if AO is structured in a wrong way that will result in the code generated with syntax error. The future work to fix this issue is to development a new component which it checks the syntax of the generated code and if there is an syntax appears it will automatically alert user about which part of AO might cause the problem and even with suggestions for solving the problem.

4) Built-in Deployment

Code Deployment module is separate from CGE at the moment. When code generated through the CGE, there is a need for it to be processed by the Code Deployment if user wants to deploy the generated code to the web server. However, it shows some kinds of inconvenient by concerning of user experience. The future work for refine the code deployment module may build it within the scope of CGE and allows user to do the initial setting (specify location of deployment folder) before actual deployment. Then once user generated code using CGE, the generated code will then CGE can automatically deployed the code to the user specified folder and access through the internet.

5) Security Issue

Security issue arises when user wants to change the service by modifying the AO. Any evil attempt by changing service to hack the system should not be allow.

Authentication and authorization are necessary here in order to maintain the system in the good form. On the other hand, there is another solution to providing a ‘sand box’ facing the user that any change user make within this sandbox only related to himself and won’t make any changes outside the sandbox. This sandbox can also be a possible safe test environment for CGE.

BIBLIOGRAHPHY

[1] http://www.patentstorm.us/patents/6260160/description.html, retrieved 18/05/2008

[2] http://www.stylusinc.com/Common/Concerns/SoftwareDevtPhilosophy.php retrieved 21/05/2008

[3] Annke Kleppe, Jos Warmer, Wim Bast, P. (2002). *MDA Explained, The Model Driven Architecture Practice and Promise*, Addison Wesley, London: p6-20.

[4] *DISSEMINATION OF GOOD PRACTICE RESULTS RAPID GROWTH AND COMPETITIVENESS THROUGH TECHNOLOGY HELSINKI* Retrieve 23/05/2008, from http://ec.europa.eu/enterprise/entrepreneurship/support_measures/docs/good-pr_helsinki_1999.pdf

[5] Martin Wirsing, Matthias H¨olzl (2006) *Software Intensive Systems* Report of the Beyond the Horizon Thematic Group

[6] http://www.linuxjournal.com/article/3508 retrieved 25/05/2008

[7]http://www.theenterprisearchitect.eu/archive/2008/01/16/mda_model_driven_architecture_ retrieved 01/06/2008

[8] Olegas Vasilecas, Diana Bugaite (2007), *APPLYING THE META-MODEL BASED APPROACH TO THE TRANSFORMATION OF ONTOLOGY AXIOMS INTO RULE MODEL* Retrieved 13/07/2008 from http://itc.ktu.lt/itc361/Bugaite361.pdf

[9] http://ksi.cpsc.ucalgary.ca/KAW/KAW96/guarino/guarino.html retrieved 26/05/2008

[10] Rudi Studer1, Stefan Decker2, Dieter Fensel3, and Steffen Staabl, *Situation and Perspective ofKnowledge Engineering* http://infolab.stanford.edu/~stefan/paper/2000/ios_2000.pdf retrieved 28/06/2008

[11] Oscar Corcho, Mariano Fernandez-Lopez, and Asuncion Gomez-Perez (2006), *Ontological Engineering: What are ontologies and How can We build Them?* Web services theory, tool, and application, Jorge Cardoso: p. 44-71.

[12] *http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/glossary-of-terms.html* retrieved 27/06/2008

[13] Martin Doerr (2006), *Web Ontology Lanauge* Web services theory, tool, and application, Jorge Cardoso: p. 96-110.

[14] *http://www.w3.org/TR/owl-guide/ access* retrieved 22/06/2008

[15]Soumen Sarkar (2005) *CODE GENERATION USING XML BASED DOCUMENT TRANSFORMATION*http://www.theserverside.com/tt/articles/content/XMLCodeGen/xmltransform.pdf retrieved 28/06/2008

[16] Soumen Sarkar (2003) *Model-Driven Programming using XSLT* http://www.codegeneration.net/articles/mdpuxslt.pdf retrieved 07/07/2008

[17] http://protege.stanford.edu/plugins/owl/api/guide.html retrieved 08/08/2008

[18] *http://saxon.sourceforge.net/* retrieved 06/08/2008

[19] Grassi, S., Barrett, S., and Sordillo, *F. 2007. Ontology based algorithm modeling: obtaining adaptation for SOA environment*. In Proceedings of the 2nd Workshop on Middleware For Service Oriented Computing: Held At the ACM/IFIP/USENIX international Middleware Conference (Newport Beach, California, November 26 - 30, 2007). MW4SOC '07. ACM, New York, NY, 18-23. DOI= http://doi.acm.org/10.1145/1388336.1388339

[20] Grassi, S.; Barrett, S., *"Dynamic Architecture Adaptation in WS Environment,"* Autonomic and Autonomous Systems, 2006. ICAS '06. 2006 International Conference on, vol., no. pp. 26--26, 19--21 July 2006.

[21] http://protege.stanford.edu/overview/protege-owl.html retrieved 07/08/2008

[22] Jack Herrington (2005), *Code Genation in Action*, Manning, p. 15-16.

APPENDIX I

This appendix is a questionaire to evaluate on the collaboration work of project.

Q: Does the Code generation engine meet your research aim?
A: The CGE fulfill the requested requirements, the architecture of the software is object oriented, easy to understand and modify. The more important aspect is that was respected the indipendence of the CGE code from the input ontologies, that are used to generate the output code. Apart from relevant or major improvement is possible to generate code for a new programming language without touching the CGE code.

Q:What do you use CGE and for what?
A: I use it on a daily basis during my test about generation of code from a model of an algorithm. In the next months it will be used for my test and will be one of the main element of my future research.

Q:What kind of features you concern about in term of code generation using ontology-based modeling algorithm?
A:The main feature is to be able to generate valid code using exactly the same model of the algorithm. All the complexity of the programming languages is stored in the system dependent ontology, where XSL rules are stored to enable the CGE to generate proper code. The concert is to include in the abstract algorithm model all the semantic information needed to decide how to generate proper code, without adding information relative to a specific language in the abstract model. This separation allows the CGE to be itself independent from the generated code and to use a common algorithm model for any system in use.

Q:What benefits you gain from the implementation of CGE ?
A: The CGE is a fundamental tool to run practical experiments about my research project, and will be a core part of a publication about those experiments. Starting with the result of Liang project will speed-up my research, having to modify and improve a well architected and well tested software.

Q: What are possible suggestions you have for CGE as future work?
A: The CGE can be improved depending on the extension in my original research project. The main is to include in the CGE the deployment procedure, now implemented as a make functionality, external from the main CGE core.
A second important improvement is the ability of the CGE to send feedback to the source of input in case of syntax errors in the generated code, associating the error with the elements of the algorithm model.

Q:How do you feel about collaboration work with Liang Shan during the summer? Any difficulty encounted? Are you satisfied with what you have achieved ?

A: The work was planned and understood in time, Liang spent a proper amount of time reading documentation and understanding the main objective of my original research. Then he was quick and effective in planning the architecture of his own project, the CGE. The development was smooth, we were in constant contact with
regular meeting to be sure to understand any problem immediately and proceed without losing time.
So I consider the collaboration very successful, in terms of the result and in terms on how was managed during the months. Liang was quick and effective in implementing and doing modification following the feedback from tests.

APPENDIX II

This appendix is the overall project schedule for last couple of month

March 10th -25th   First meeting, introduction about Simone's research project, aim and objectives. Read a small set of papers and did a meeting to understanding what was the proposal project with some level of details.

March 25th - April 25th, did background research on the technologies to be used in the development of the project. Framework and platform to be used for the code and checked the presence of the needed libraries that was going to be used for known technologies like XSL and OWL.

April 26th - May 26th Develop a first version test appliation for traversing ontology tree and retrieve one by one the nodes of the Syntax Tree. The OWL libraries (from the Protege software, version 3.4 beta) was tested and used to achieve this step.

May 26th - June 15th Develop the first verion of Code generation engines, adding the real code generation to the visit of the Syntax Tree. Saxon 6.6.5 libraries was tested and used. A few problems were encounted and fixed doing ad-hoc tests.

June15th - August 1st From test and evaluation of the previous version a second version of CGE was developed including all the needed requirements. Some automation was added and it was improved to cover all the original requirements.

August 1st - Sep Test and Evaluation continued with some bug fixing and the addition of the Make feature and a graphical user interface to have a first degree of user friendly interface