Maintaining Consistency between Natural Language Specifications and Software Architectures

Supervisor:	Dr. Joseph Kiniry
Subject Area:	Software Engineering
Pre-requisite:	Good knowledge of the Java programming language
Co-requisite:	Deep knowledge of the Grammatical Framework (GF) and the BON specification language will be obtained during the course of this project
Subject Coverage:	BON, GF compiler design and implementation and type-logical grammars
Project Type:	Design & Implementation
Hardware:	PC or workstation running nearly any mainstream OS

Description

BON[1,2] is a high-level specification language for describing structured systems, including programs, web sites, databases, etc. The initial step of the BON method is to create a static model of the system being specified. The static model outlines the classes making up the system, their interfaces, how they are related to each other and how they are grouped into clusters of the system. This project focuses on the creation of the BON static class model.

The first step in the creation of the class model is the creation of an informal class chart. An informal class chart is written in structured natural language. Each class chart defines the queries and commands (i.e., the class's API) and constraints (its invariants) of a class.

Class charts are then translated into formal static diagrams with typing and software contracts. The BON notation used in the creation of these diagrams has two variants: graphical BON and textual BON. This project focuses on the textual BON notation.

Currently the translation of the informal charts into the formal diagrams causes problems. Due to ambiguity in natural language, informal charts often lack clarity. Moreover, different developers describe the same class in different ways. This project focuses on this ambiguity problem by defining a subset of natural language that has a precise semantics. If a developer writes documentation using this subset, then automatic translation of the natural language documentation to formal static diagrams is possible.

The Grammatical Framework[4] (GF) is a formal language for **multilingual grammar** applications. A GF specification is called a grammar which defines a language. Dozens of languages have been analyzed with GF[3,5]. From this definition, the following language processing components are derived:

- **parsing**: to analyse a language described by the grammar
- **linearization**: to generate from a parse tree the concrete syntax of a language
- **translation**: to analyse (parse) one language and generate (linearize to) another

For example, GF includes a natural language processing framework and has been used in authoring and translating of English software specifications to and from OCL[6].

The primary goal of this project is to build a tool which automatically performs the translation between informal class charts written in restricted natural language to class models written in formal BON notation. The tool that will perform this translation is called the ESC/IBON (Extended Static Checker/Informal BON) tool. It will be realized by a plugin for the Eclipse IDE and written in the Java programming language. The ESC/IBON tool will analyse informal class

charts and perform the translation to formal BON through the use of the GF library. A GF grammar will be defined specifically for translating natural language into textual BON assertions. Our goals is to assist the BON software process by providing developers with support for writing unambiguous software descriptions. Furthermore, we expect that this tool will reduce the maintence cost associated with having the same class declaration written in two seperate places, as changes will only need to be made once to the informal charts.

Mandatory

1. Learn BON and GF.
2. Build experience with the Java compilers, libraries, IDEs, the BON language and GF.
3. Review existing Javadoc documentation from major Java vendors (e.g., Sun, the Apache Foundation, etc.) to characterize and categorize the standard grammatical patterns developers use when writing simple English specifications.
4. Using GF, define a type-logical grammar, G_f, for the restricted form of English used in informal BON diagrams and the aforementioned Javadoc documentation.
5. Use the GF toolkit to use G_f to produce an implementation capable of transforming those restricted English sentences found in informal BON features into types expressed in formal BON.
6. Precisely describe the refinement relation between G_f and function types in BON's type system.

Discretionary

1. Write a component that translates a restricted form of formal BON specifications into English using the type-logical grammar and refinement defined in the mandatory requirements.
2. Define a second type-logical grammar, G_c for a restricted form of English for expressing assertions expressible in formal BON.
3. Use the GF toolkit to use G_c to produce a component capable of transforming restricted English expressions found in Informal BON constraints into formal BON assertions.
4. Write a static checker for either or both restricted forms of English corresponding to grammars G_f and G_c to give early, rapid feedback to developers about the quality and utility of their documentation.

Exceptional

1. Perform a similar analysis and tool building to convert from full/arbitrary formal BON types and formal BON assertions into English.

Sources of information and preparatory reading

1. Kim Walden. *Business Object Notation* http://www.bon-method.com/handbook_bon.pdf
2. Kim Walden and Jean-Marc Nearson. *Seamless Object-Orientated Software Architecture* http://www.bon-method.com/book_print_a4.pdf
3. Kristofer Johannisson. *Formal and Informal Specifications* http://www.cs.chalmers.se/~krijo/thesis/thesisA4.pdf
4. The Grammatical Framework http://www.cs.chalmers.se/~aarne/GF/
5. Aarne Ranta. *Grammars as Software Libraries* http://www.cs.chalmers.se/~aarne/articles/libraries-kahn.pdf
6. Object Constraint Language Specification, version 2.0 http://www.omg.org/technology/documents/formal/ocl.htm